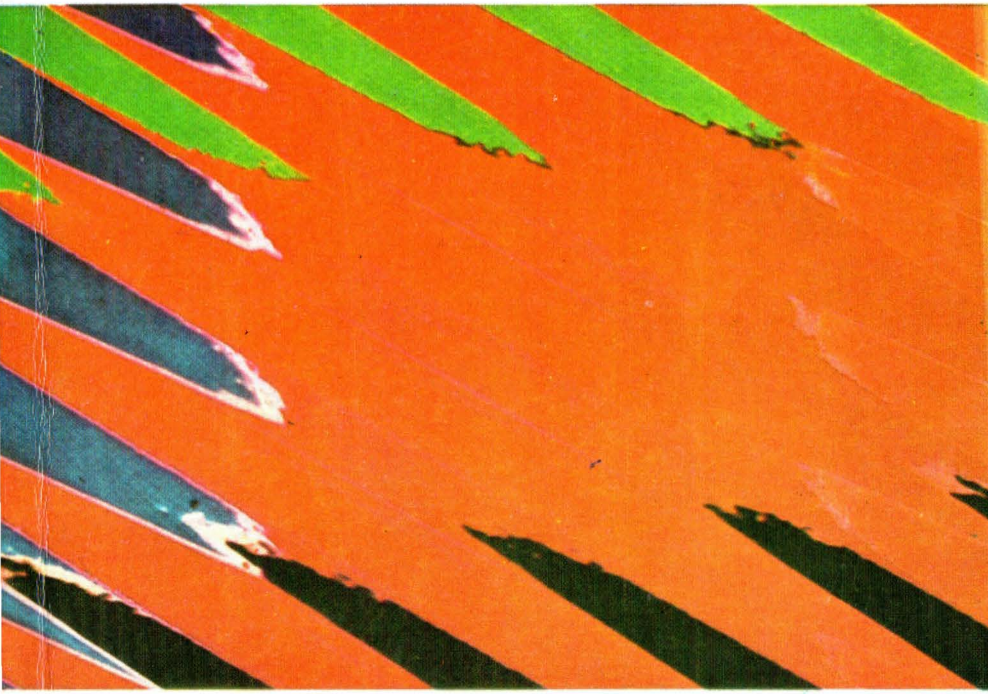
Apple Merlin
使用手冊



馬行飛 譯

電 腦 科 技 社

Apple Merlin
使 用 手 冊

馬行飛 譯

Retro Workshop
果粉工作室

果粉藏書

電 腦 科 技 社

出版·印刷：
電　腦　科　技　社
澳門黑沙灣台來街821號
定　價：

目　　錄

1.　前　　言 ………………………………………… 1

1.1.　組合語言的源由 ………………………………… 1
1.2.　背景以及特性 …………………………………… 3
1.3.　建議讀物 ………………………………………… 4

2.　系統所需的設備 ……………………………… 7

2.1.　可與硬體配合之一覽表 ………………………… 7

3.　初學者指南 …………………………………… 9

3.1.　簡　介 …………………………………………… 9
3.2.　輸　入 …………………………………………… 10
3.3.　系統及輸入命令 ………………………………… 13
3.4.　組合程式 ………………………………………… 16
3.5.　儲存和執行程式 ………………………………… 18
3.6.　MERLIN備份磁片的複製 ……………………… 20

4.　執 行 模 式 …………………………………… 21

4.1.　C：CATALOG（磁碟目錄）……………………… 21
4.2.　L：LOAD（載入檔案）………………………… 22
4.3.　S：SAVE（儲存檔案）………………………… 22
4.4.　A：APPEND（附加檔案）……………………… 23

4.5. D:DRIVE（磁碟機編號）……23
4.6. E:EDITOR（編輯器）……23
4.7. O:SAVE OBJECT CODE（儲存目的碼）……23
4.8. Q:QUIT（結束）……24
4.9. R:READ（讀取文字檔案）……24
4.10. W:WRITE（寫入文字檔案）……25

5. 編 輯 器……27

5.1. 命令模式……27
5.1.1. HImem……28
5.1.2. NEW……28
5.1.3. PR#（0-7）……28
5.1.4. USER……28
5.1.5. TABS……28
5.1.6. LENgth……29
5.1.7. Where……29
5.1.8. MONitor……29
5.1.9. TRuncON……30
5.1.10. TRuncOFf……30
5.1.11. Quit……30
5.1.12. ASM……30
5.1.13. Control-D……31
5.1.14. Delete……31
5.1.15. Replace……31
5.1.16. List……32

5.1.17. .（句點）…… 32
5.1.18. /（列號）…… 32
5.1.19. Print …… 33
5.1.20. PRinTeR（命令）…… 33
5.1.21. Find …… 34
5.1.22. Change …… 34
5.1.23. COPY …… 35
5.1.24. MOVE …… 35
5.1.25. Edit …… 35
5.1.26. Hex-Dec Conversion …… 36
5.1.27. TEXT …… 36
5.1.28. FIX …… 36
5.1.29. SYM …… 36
5.1.30. VIDeo …… 37
5.1.31. FW（尋字）…… 38
5.1.32. CW（換字）…… 38
5.1.33. EW（改字）…… 38
5.1.34. VAL …… 39
5.2. 附加／插入模式 …… 39
5.2.1. 附加(Add) …… 39
5.2.2. 插入(Insert) …… 40
5.2.3. Control-L …… 40
5.3. 編輯模式 …… 40
5.4. 編輯模式的命令 …… 41
5.4.1. Control-I（插字）…… 41
5.4.2. Control-D（刪字）…… 41

5.4.3. Control-F（尋字）…… 41
5.4.4. Control-O（插入特殊字元）…… 41
5.4.5. Control-P（重複＊＊＊）…… 42
5.4.6. Control-@（首尾＊＊）…… 42
5.4.7. Control-C或Control-X（取消）…… 42
5.4.8. Control-B（跳至列首）…… 43
5.4.9. Control-N（跳至列末）…… 43
5.4.10. Control-R（回復）…… 43
5.4.11. Control-Q（刪除游標後字元）…… 43
5.4.12. Return（RETURN鍵）…… 43

6. 組 合 器 …… 45

6.1. 數值格式 …… 45
6.2. 原始碼格式 …… 46
6.3. 數學式 …… 47
6.4. 立即資料 …… 48
6.5. 定址模式（6502運算碼）…… 49
6.6. SWEET 16運算碼 …… 50
6.7. 虛運算碼一指令 …… 51
6.7.1. EQU（＝）…… 51
6.7.2. ORG …… 51
6.7.3. OBJ …… 52
6.7.4. PUT …… 52
6.7.5. VAR …… 53
6.7.6. SAV …… 53
6.7.7. DSK …… 54

6.7.8. END …… 54
6.8. 定格式 …… 54
6.8.1. LST ON/OFF …… 54
6.8.2. EXP ON/OFF …… 55
6.8.3. PAU …… 55
6.8.4. PAG …… 56
6.8.5. AST …… 56
6.8.6. SKP …… 56
6.8.7. TR …… 56
6.9 字串 …… 57
6.9.1. ASC …… 57
6.9.2. DCI …… 57
6.9.3. INV …… 58
6.9.4. FLS …… 58
6.9.5. REV …… 58
6.10. 資料和空間配置 …… 58
6.10.1. DA …… 58
6.10.2. DDB …… 59
6.10.3. DFB …… 59
6.10.4. HEX …… 60
6.10.5. DS …… 60
6.10.6. KBD …… 60
6.10.7. LUP …… 60
6.10.8. CHK …… 61
6.10.9. ERR …… 62
6.10.10. USR …… 62

6.11. 條件組合 …… 65
6.11.1. DO …… 65
6.11.2. ELSE …… 66
6.11.3. FIN …… 66
6.12. 集體指令 …… 67
6.12.1. MAC（MACro集體指令）…… 67
6.12.2. PMC（>>>）…… 67
6.13. 變數 …… 68

7. 集體指令 …… 69

7.1. 定義集體指令 …… 69
7.2. 巢狀式的集體指令 …… 70
7.3. 特殊的變數 …… 71
7.4. 程式範例 …… 73
7.5. 集體指令程式庫（Macro Library）…… 74

8. 技術資料 …… 75

8.1. 一般技術資料 …… 76
8.2. MERLIN記憶體配置圖（RAM CARD版）77
8.3. MERLIN記憶體配置圖（48K版）…… 78
8.4. 符號表(SYmbol Table) ……
8.5. 移位鍵SHIFT KEY在MERLIN下的用法 79
8.6. 80行卡（8Ø COLUMN BOARD）的用法 80
8.7. Configure ASM程式說明 …… 80
8.8. 錯誤信息 …… 81
8.8.1. BAD OPCODE（錯誤的運算碼）…… 81

8.8.2. BAD ADDRESS MODE（錯誤的定址模式）81
8.8.3. BAD BRANCH（錯誤的分歧）……… 82
8.8.4. BAD OPERAND（錯誤的運算元）… 82
8.8.5. DUPLICATE SYMBOL（符號重複定義）82
8.8.6. MEMORY FULL（記憶體已滿）…… 82
8.8.7. UNKNOWN LABEL（標題未定義）‥ 82
8.8.8. NOT MACRO（不是集體指令）…… 82
8.8.9. NESTING ERROR（巢次錯誤）…… 83
8.8.10. BAD "PUT"（不正確）……………… 83
8.8.11. BAD "SAV"（錯誤）……………… 83
8.8.12. BAD INPUT（輸入錯誤）…………… 83
8.8.13. BREAK（中止）……………………… 83
8.8.14. BAD LABEL（標題錯誤）…………… 83
8.9. 特別注意事項－記憶體已滿錯誤…………… 84

9. SOURCEROR ………………………………… 87

9.1. 簡介………………………………………… 87
9.2. 使用 SOURCEROR ……………………… 88
9.3. 反組合命令………………………………… 89
9.4. 命令描述…………………………………… 90
9.4.1. L（List）…………………………… 90
9.4.2. S（SWEET）……………………… 90
9.4.3. N（Normal）……………………… 90
9.4.4. H（Hex）…………………………… 90
9.4.5. T（Text）…………………………… 91
9.4.6. W（Word）………………………… 91

9.5. 內部命令(Housekeeping Commands) ·· 92
9.5.1. / (Cancel 取消) ······ 92
9.5.2. R (Read) ······ 92
9.5.3. Q (Quit) ······ 93
9.6. 最後處理程序 ······ 93
9.7. 完成後檔案的處理 ······ 94
9.8. 記憶已滿訊息 ······ 95
9.9. LABELER 程式 ······ 96
9.10. LABELER 的命令 ······ 96
9.10.1. Q:QUIT ······ 96
9.10.2. L:LIST ······ 96
9.10.3. D:DELETE LABEL(S) ······ 97
9.10.4. A:ADD LABEL ······ 97
9.10.5. F:FREE SPACE ······ 97
9.10.6. U:UNLOCK SRCRR.OBJ ······ 97

10. SWEET 16簡介 ······ 99

10.1. 列表# 1 ······ 104
10.2. 列表# 2 ······ 105
10.3. 列表# 3 ······ 105

11. SWEET 16一個虛擬的16位元微處理機 107
11.1. 簡介 ······ 107
11.2. 指令介紹 ······ 109
11.3. SWEET 16 運算碼總說 ······ 110
11.3.1. 暫存器指令 ······ 110

11.3.2. 非暫存器指令(ea:爲有效位址，+/－127)…110
11.4. 暫存器指令………………………………………… 111
11.4.1. SET 設定………………………………………… 111
11.4.2. LOAD 載入………………………………………… 111
11.4.3. STORE (儲存) ……………………………… 111
11.4.4. LOAD INDIRECT 間接載入……… 112
11.4.5. STORE INDIRECT (間接儲存)… 112
11.4.6. LOAD DOUBLE-BYTE INDIRECT
(間接雙位元組載入)…………………… 113
11.4.7. STORE DOUBLE-BYTE INDIRECT
(間接雙位元組儲存)…………………… 113
11.4.8. POP INDIRECT (間接取出)……… 114
11.4.9. STORE POP INDIRECT (間接儲存) 114
11.4.10. ADD (加)…………………………………… 115
11.4.11. SUBTRACT (減) ……………………… 115
11.4.12. POP DOUBLE-BYTE INDIRECT
(間接雙位元組堆入)……………………… 116
11.4.13. COMPARE (比較)……………………… 117
11.4.14. INCREMENT (加1)…………………… 117
11.4.15. DMCREMENT (減1)…………………… 118
11.5. 非暫存器指令………………………………… 118
11.5.1. 回到 6502 工作模式 ……………………… 118
11.5.2. 無條件分歧………………………………… 118
11.5.3. 無溢位(爲零)則分歧……………… 119
11.5.4. 溢位則分歧………………………………… 119

11.5.5. 正數則分岐 …… 120
11.5.6. 負數則分岐 …… 120
11.5.7. 爲零則分岐 …… 120
11.5.8. 不爲零則分岐 …… 121
11.5.9. 負1則分岐 …… 121
11.5.10. 非負1則分岐 …… 121
11.5.11. 中斷(BREAK) …… 121
11.5.12. 從 SWEET 16的副程式返回 …… 121
11.5.13. 呼叫 SWEET 16的副程式 …… 122
11.6. 工作原理 …… 122
11.7. RTS如何成爲JSR? …… 123
11.8. 運算碼的副程式 …… 124
11.9. 記憶體配置 …… 125
11.10. 使用者自行修改 …… 125

12. APPLESOFT的原始程式 …… 127

12.1. SOURCEROR.FP …… 127
12.1.1 產生Applesoft反組合原始程式的步驟 128

13. 語彙解釋 …… 131

14. 程式範例 …… 135

14.1. 浮點運算(Floating Point Routines) …… 135
14.2. 乘/除運算(Multiply /Divide Routines) 135
14.3. PRDEC 顯示10進位數 …… 136
14.4. MSGOUT顯示資料 …… 136
14 UPCON轉爲大寫 …… 136
1 6. GAME PADDLE Printer Driver …… 136

15. 公用程式 …………………………………… 139

15.1. FORMATTER定格式 ………………………… 139

15.2. CHRGEN 70 …………………………………… 140

15.3. XREF, XREF.XL,和STRIP……………… 141

15.3.1. MERLIN符號表列印範例 ……………… 142

15.3.2. MERLIN XREF列印範例 ………… 142

15.3.3. XREF使用說明 ……………………… 143

15.3.4. 使用XREF時注意事項 ……………… 144

15.3.5. XREF.XL 使用說明 ………………… 145

15.3.6. 使用XREF.XL 注意事項 …………… 145

15.4. STRIP去掉註解 ……………………………… 146

15.5. PRINTFILER ………………………………… 147

15.5.1. 應用 …………………………………… 147

15.5.2. 如何使用PRINTFILER ……………… 148

15.5.3. 修改PRINTFILER …………………… 148

15.5.4. PRINTFILER的比較 ………………… 149

15.5.5. 修改PRINTFILER 的選擇 …………… 149

1. 前　言

1.1 組合語言的源由

也許某些人會問“什麼是組合語言？”或者“爲什麼我要用組合語言？BASIC已經很好了！”；然而在這裡，我們並沒有多餘的篇幅來對這個問題，做有系統的闡述，我只試著簡單地回答上述這些問題。

電腦語言通常區分爲高階（High level）和低階（Low level）兩種語言。BASIC，COBOL，FORTRAN和PASCAL都是高階語言。高階語言是使用類似英文字句（命令），而且要經過多層階段的翻譯或編譯，才能眞正地放在記憶體中執行。這些過程中所耗費的時間，往往造成BASIC和其他高階語言執行起來遠比相同的組合語言爲慢的原因。此外，高階語言通常也比較浪費記憶體。

基本上，你的電腦只懂得兩件事情，關（OFF）及開（ON）。所有的計算都是以加法或減法來處理，但是都以極快的速度進行。電腦所能了解的數值系統是以2爲基底（二進位系統）。“1”是以 ØØØØØØØ1 來表示，“2”則爲

ØØØØØØ1Ø 。

6502微處理器(microprocessor)的算術運算單元(ALU)中，具有五個8位元(bit)的暫存器和一個16位元的暫存器。所有的資料最後都是經由這些暫存器來處理。即使是這種最低階的碼，也需要一個程式來使得6502正確地操作。這個程式是在硬體上已經連結在6502的內部裏。這個微程式分做三個週期來操作。首先它從RAM記憶體中拿取(Fetch)指令，然後加以解碼(Decode)，最後再去執行(Execute)該指令。

指令是以一個、二個或三個位元組(Byte)爲單位儲存在記憶體中。一個位元組包含8個位元(Bit)，通常是以十六進位(16爲基底)的型式來代表。某些早期的微計算機是由其前置面板的16個開關來輸入資料。每個開關的開或關，結合起來就成爲十六進位的資料。這種裝置需要一個電腦程式使得十六進位的資料分爲兩個8位元的資料，如此6502才可以了解。

更高一層的電腦(依舊需要更多的程式控制)，使用者可以三個文字一組的“助憶符號”來輸入他的資料。助憶符號是一種碼，它所組成的文字和微處理器的操作互相關連。例如LDA表示“LoaD the Accumulator”(載入累積器)。標準的Apple Ⅱ有一個內建的迷你組合器，能允許設計一些簡單的組合語言。

但是即使如此，並不足以產生一個巨大而且可以了解的程式。除了能使用三個字元的助憶符號外，一個完全的組合器還可以讓程式師使用“標題(Label)”。標題是表示程式中特別的一段，但是目前在記憶體中還未定義。此外，組

合器還需要類似 BASIC 程式中的列號儲備功能，這種功能可以允許程式師在程式中插入幾列以及其他的編輯功能。這些功能MERLIN都可以達到。

最後，一個高階語言，像BASIC ，本身也是由組合語言所構成。但是它可以使用一個命令，例如 PRINT ，並且縮減爲一個位元組的記號（ Token ）儲存在記憶體中。

在使用MERLIN或其他的組合器之前，希望使用者能熟知 6502 的結構、定址模式等等。本使用手册沒有介紹組合語言程式設計，使用者可以在書局買到許多有關 6502 組合語言程式設計方面的書，本節之後會提到一些以供參考。

1.2 背景以及特性

MERLIN 是一個以“TED”爲準的編輯組合器。這意味著MERLIN幾乎可以全部和TED Ⅱ + 的命令助憶符號，虛運算碼等等相吻合。

最初的TED ASM 是由 Randy Wiggington和Gary Shannon 所撰寫的。它以不同的版本，廣泛地分佈到不同的使用者和集團中。每個使用者表面上可以增加他自己的改進措施。這點MERLIN也不例外。但是MERLIN 更進一步加上了Macro（集體指令）的功能，使得MERLIN成爲目前Apple Ⅱ上最進步和複雜的編輯組合器。但是MERLIN依舊保留了所有TED容易使用的特性，使得組合語言的初學者覺得很適宜。

除了集體指令以外，MERLIN 比較明顯的改變如：利用邏輯運算子（ operator ）AND, OR, 與 EOR ，數學運算子

的除法，列印或不列印列號（ line number ）的能力，與更快速的編輯功能等等。同時，編輯模組也包括了許多新增的命令以輔助編輯，而且“ Read ”命令可以將 Apple Ⅱ 的文字檔（ text file ）讀入編輯緩衝區（ buffer）內，因此，MERLIN 也可以處理由其他組合器（例如：DOS Tool Kit ）所產生的原始檔案。

MERLIN 假設你的系統至少有 48K的記憶體，而且在 DOS 3.3之下作業。特別留意修改過而不是標準的DOS(custom DOS)，MERLIN 是無法使用的，在開始執行後，MERLIN自動設MAXFILES為 2 ，當離開後，又自動還原為原先的值。

1.3 建議讀物

SYSTEM MONITOR - Apple Computer, Inc. Peeking at Call-Apple, Vol I.

APPLE II MINI-ASSEMBLER - Apple Computer Inc. Peeking at Call-Apple Synertek Programming Manual., Synertek 6500-20.

PROGRAMMING THE 6502 - Rodnay Zaks, Sybex C-202.

THE APPLE MONITORS PEELED - WM. E. Dougherty, Apple Computer, Inc.

A HEX ON THEE - Val J. Golding, Peeking at Call-Apple, Vol. II.

FLOATING POINT PACKAGE - Apple Computer, Inc., The Wozpak II

FLOATING POINT LINKAGE ROUTINES - Don Williams, Peeking at Call-Apple Vol I

APPLE II REFERENCE MANUAL - Apple Computer, Inc.

ASSEMBLY LINES - by Roger Wagner

在SOFTALK雜誌中，一系列爲初學者介紹的文章，對於想要很容易學會組合語言的初學者，是極有價值的介紹。

ASSEMBLY LINES: THE BOOK - by Roger Wagner

是Assembly Line序列中最初18種出版物的精編本。本文中廣泛地加以編輯，最後還加上一個類似百科全書的附錄。這個附錄不僅說明每個6502命令的詳細內容，並且簡潔地以圖表似的條例來討論它的用途。

CONVERTING BRAND X TO BRAND Y - by Randall Hyde
Apple Orchard, Volume 1, No.1, March/April 80.

對各種組合器間的轉換，是很有用的參考。

CONVERTING INTEGER BASIC PROGRAMS TO ASSEMBLY LANGUAGE
by Randall Hyde
Apple Orchard, as above.

HOW TO ENTER CALL - APPLE ASSEMBLY LANGUAGE LISTINGS
Call-APPLE, Volume IV, No.1, January 81.

MACHINE TOOLS
Call-APPLE in Depth, No. 1

2. 系統所需的設備

* 48K Apple Ⅱ
* 16K RAM CARD
* 80 COLUMN BOARD（可選擇）
* LOWER CASE BOARD（可選擇）

2.1 可與硬體配合之一覽表

* VIDEX VIDEOTERM
* FULL-VIEW 80-80 COLUMN BOARD
* M & R SUP'R'TERMINAL 80COLUMN BOARD
* ALS SMARTERM 80 COLUMN BOARD
* VISTA VISION 80-80 COLUMN BOARD
* OMEGA MICROWARE RAMTEX 16-16K RAM BOARD
* ANDROMEDA 16K BOARD
* MICROSOFT 16K RAM BOARD

* WIZARD 80-80 COLUMN BOARD

注意：MERLIN和上述表中的硬體配合，都已經測試過。對於未列入表中的80行卡，作者並不保證。

3. 初學者指南

注意事項及範例是針對於MERLIN的初學者

3.1 簡 介

本節的目的並非提供在組合語言程式設計時的指令，而是很廣泛地介紹 MERLIN 初學組合語言程式設計的程式師們，以及MERLIN的特殊用途。

注意：在本手册中，“模式(mode)”和“模組(module)”有不同的用法。

本節中，“模組”代表組成MERLIN不同的電腦程式。MERLIN包含了有四個模組：

1. The EXECUTIVE (執行器)
2. The EDITOR (編輯器)
3. The ASSEMBLER (組合器)
4. The SYMBOL TABLE GENERATOR (符號表產生器)

每一個模組是被下列兩組控制模式之一所組成：

1) The EXECUTIVE，簡稱EXEC，它的標記為'%'。

2) The EDITOR，標記爲 ':'。

EXECUTIVE CONTROL MODE
　　Executive Module （執行模組）

EDITOR CONTROL MODE
　　Editor Module （編輯器模組）
　　Assembler Module （組合器模組）
　　Symbol Table Generator Module
　　（符號表產生器模組）

"模式"可用來指示目前的控制模式（由系統標記來指示）或者是當在控制模式時，隨後要發出的輸入命令，此時系統被稱爲是在"［輸入命令］模式"。例如：當發出ADD命令來輸入程式時，系統被稱爲在"附加模式"。

結束［輸入命令］模式會使系統回到控制模式。

3.2 輸　入

熟悉組合語言或高階語言的程式師，都知道輸入必須規定格式，也就是：標題、運算碼、運算元和說明，這些必須要輸入在特定的欄位內，否則就不會被組合器所承認。

在MERLIN系統內，TAB這個運算子（operator）提供了半自動的定格式特性。

當輸入程式時，要記得在組合期間，每一個原始碼中的空白會導致一個表格定位到下一個欄位的起始點上。讓我們輸入下列的短程式作爲範例說明：

步驟從最先開始：

1. BRUN MERLIN
2. 當%標記出現在執行模式功能表的底部時，鍵入

'E'，系統立即會進入編輯器控制模式。

3. 由於我們是要輸入一個全新的程式，所以在':'標記出現後，鍵入'A'，然後鍵入'RETRUN'。這時，一個'1'會出現在下一列，其右有一個游標會自動地將一個空白定位到列號的右邊一格。'1'和以後所有的列號都會在鍵入RETURN後出現，就好比在BASIC 中的列號一樣。不同者，在原始碼中，列號並不是用作跳躍(JUMP)至副程式或者類似GOTO的敍述句。
4. 在列號1輸入一個星號'*'。任一列中。星號出現在第一個字元，表示該列就和BASIC 中的REM 敍述句是一樣的——它告訴組合器這是一個註解列，在星號後面可輸入任何字。爲了確定這一點，鍵入'DEMO PROGRAM 1'，然後鍵入RETURN。
5. 在RETURN之後，游標又再度向下一列，然後出現'2'，游標向右跳一個空白。
6. 現在，鍵入空白鍵，然後鍵入'OBJ'，再接著一個空白，鍵入'$3ØØ'，最後鍵入RETURN。注意：在大多數的情況下，'OBJ'這個虛運算碼並不需要。
7. 在第3列，依照相同的程序：空白，鍵入'ORG'，空白，鍵入'$3ØØ'，最後鍵入RETURN。
8. 在第4列，在列號的後面不再鍵入空白，而鍵入BELL'，空白，'EQU'，空白，'$FBDD'，RETURN。
9. 在第5列鍵入'START'，空白，'JSR'，空白

，'BELL'空白，';'，'RING THE BELL'。分號（；）通常在傳統上是用來表示註解開始的標誌。

10. 第6列鍵入'END'，空白，'RTS'，RETURN。

11. 程式已經輸入完畢，但是系統還是在ADD模式。要離開ADD模式，只要鍵入RETURN或者CTRL-X, RETURN。系統標記':'又會重新出現在螢幕的左邊，表示系統已經回到控制模式。

12. 這時，螢幕上應該像下列一樣：

```
1  *DEMO PROGRAM 1
2           OBJ   $3ØØ
3           ORG   $3ØØ
4  BELL     EQU   $FBDD
5  START    JSR   BELL          ;RING THE BELL
6  END      RTS
```

注意每一個字串都被移動到特定的欄位去。不包括列號，這裏有四個特定的欄位。

欄位#：

第一個是預留給標題。BELL，START和END都是標題。

第二個是預留給運算碼，例如MERLIN的虛運算碼，如OBJ，ORG，和EQU，以及65Ø2的運算碼，例如JSR和RTS。

第三個是預留給運算元，例如$3ØØ，和BELL（在這個情況下）。

第四個是包含了一些程式註解。

在這個範例中可以很明顯地看出，在原始碼中並不需要

輸入多餘的空白鍵，來達到定格式的目的。

大略地說，在列號的後面：

1) 不要鍵入空白表示標題。在標題後面的空白（如果沒有標題，就是列號的後面）是運算碼的欄位。
2) 在運算碼後的空白是運算元的欄位。在運算元後的空白是用作註解的欄位。如果沒有運算元，那就鍵入空白和一個分號。

3.3 系統及輸入命令

MERLIN擁有一個強有力而複雜的內建（built-in）編輯器。在操作的功能也許是很複雜，但經過一些練習後，你會發現事實上是很容易使用的。

在下面的段落中，僅包括了輸入命令用法的簡短說明。所有的系統及輸入命令都是在編輯器控制模式下使用，它的標記為'：'。

當附加或插入一列時，若以CTRL-X，CTRL-C 或是 RETURN作為一列的第一個輸入字元時，都會離開現在的［輸入命令］模式，並且系統回到控制模式。若在編輯某列時，CTRL-X或CTRL-C 會離開編輯模式，然後系統回到控制模式。

其他的系統及輸入命令都會自動地結束或者在鍵入RETURN後結束。

在原始碼中插入或刪除某列的操作非常簡單。下列的例子將插入三列在第 4 列及第 5 列之間。

1. 在標記 ':' 出現後，鍵入 'I'（插入），數值 '5'，然後鍵入RETURN。所有插入的新列，都會從指定的列號開始。
2. 輸入一個星號，然後鍵入RETURN。注意插入模式並未結束。
3. 重複步驟2。
4. 輸入一個空白，鍵入 'TYA' 和RETURN。

此時在螢幕上，應該跟下列的相同：

```
:I5
   5*
   6*
   7      TYA
   8
```

5. 鍵入RETURN 。然後系統回復到控制模式。
6. 將原始碼條列出來。

```
:L
     1  *DEMO PROGRAM 1
     2          OBJ    $3ØØ
     3          ORG    $3ØØ
     4  BELL    EQU    $FBDD
     5  *
     6  *
     7          TYA
     8  START   JSR    BELL        ;RING THE BELL
     9  END     RTS
```

新的三列（ 5, 6 和 7 ）已經插入在原始碼中，並且隨後的原始碼（現在的第8、9列）已經重新編號過。

使用刪除命令也是同樣的簡單。

1. 在控制模式下，輸入'D5'以及RETURN。此時在螢幕上不會出現什麼。
2. 條列原始碼。原始碼列表會比原來的少了一列。僅包括一個星號的該列消失了，而且隨後的列號都重新加以編號過。

也有可能以一個步驟來刪除一段區域。

1. 在控制模式下，輸入'D5,6'及RETURN。
2. 條列原始碼。

在上例中的第5列及第6列(包括一個星號以及TYA運算碼)會被刪除掉，而且也將隨後的列號加以重新編號。條列後的結果和輸入一節中的步驟13相同。

自動重新編號的特性，使得你在刪除時，要注意從最高的列號開始刪除，然後到最低的列號爲止。

附加，插入或編輯命令都有一些由控制字元組成的次命令。以BELL程式來做個說明：

1. 在標記':'出現後，輸入'E'(編輯命令)，然後跟著一個列號…(使用'6'作爲示範)，最後鍵入RETURN。於是第6列會以規定的格式出現在螢幕的下一列。並且游標在'END'的字元'E'之上：

```
6 END      RTS
```

2. 鍵入CTRL-D。在游標位置的字元消失了。再鍵入CTRL-D兩次，於是'END'被刪除，並且游標移

動到運算碼的左邊。

3. 鍵入 RETURN 然後列出程式。在原始碼的第 6 列，僅有列號和運算碼。
4. 重複步驟 1.。
5. 這一次，鍵入 CTRL-I。不要用空白鍵或箭頭鍵來移動游標。鍵入 'END'，然後 RETURN。
6. 列出程式，第 6 列已經恢復原狀。

如果你在編輯一個單列，鍵入 RETURN，就會回到控制模式。在上面的步驟 1.，如果你指定爲一個區域（如 'E3,6'）的話，在編輯命令後，鍵入 RETURN 會把指定區域內的下一列顯示出來。這時，你可以利用不同的次命令來選擇編輯功能。鍵入 RETURN，會進入下一列；或者以 CTRL-C 來結束編輯模式。

注意：鍵入 RETURN，會使得出現在螢幕的該列資料，完全地存在記憶體中。

其他編輯模式下次命令的操作都相類似。參閱第 3 節，然後再練習一些操作方法。

3.4 組合程式

接著使用MERLIN的下個步驟就是將原始碼組合成目的碼。

在標記 ':' 出現後，鍵入編輯模組系統命令 ASM，然後鍵入 RETURN。此時，你的螢幕會出現這樣：

```
UPDATE SOURCE (Y/N)?
```

鍵入N，然後你會看到：

```
ASM                        1    *DEMO PROGRAM  1
                           2              OBJ   $3ØØ
                           3              ORG   $3ØØ
                           4    BELL      EQU   $FBDD
Ø3ØØ     2Ø DD FB          5    START     JSR   BELL    ;RING THE BELL
Ø3Ø3     6Ø                6    END       RTS

--END ASSEMBLY--

ERRORS:  Ø

4 BYTES

 SYMBOL TABLE - ALPHABETICAL ORDER

      BELL      =$FBDD         ?    END          =$Ø3Ø3

 ?    START     =$Ø3ØØ

 SYMBOL TABLE - NUMERICAL ORDER

 ?    START     =$Ø3ØØ         ?    END          =$Ø3Ø3

      BELL      =$FBDD
```

如果不是類似上面的列表情形，系統就會響鈴而且顯示錯誤訊息，注意在錯誤訊息中所提到的列號；接著鍵入RETURN直到"…BYTES…"出現爲止。回到輸入一節，然後比較步驟13的列表。特別注意不正確欄位中的字元。

如果一切正常的話，在螢幕的右半部是很熟悉的按規格的原始碼。

在列號的左邊，從第5列開始，是一系列的數字和文字。這就是目的碼一運算碼和運算元組合爲對應的十六進位之機器語言。

由左至右，第一部分的字元是程式在記憶的起始位址（參見“虛運算碼－指令”一節中，OBJ 和 ORG的定義）。在其後是數字 '2Ø'；這是 JSR 運算碼所對應的一個位元組的機器碼。

注意：標題 'START'，並未組合成目的碼；註解，說明或虛運算碼，如 OBJ 和 ORG 也是一樣。這些組件只是讓程式師和組合器方便而已。但對電腦本身並無用處，因此並不會被轉換成機器語言。

第 5 列接下來的兩個位元組（每一對字元是一個位元組）是和第 4 列的運算元有類似的外觀。看一下，在第 4 列中，我們告訴組合器，標題 'BELL' 相等於位址 $FBDD。在第 5 列中，當組合器碰到 'BELL' 運算元時，它就會以指定的位址來取代。由於 65Ø2 微處理器的慣例，高位元組和低位元組的次序是相反的。

其他出現的資料是來解釋組合器本身的，原始碼中的錯誤爲零，而且有四個位元組的目的碼（計算位址以後的位元組數）。

3.5 儲存和執行程式

使用MERLIN的最後一個步驟就是去執行程式。在此之前，最好先把原始碼儲存起來。儲存目的碼則必須在組合無誤後才用。

1. 如果需要的話，回到控制模式；然後鍵入 'Q' RETURN。系統會離開編輯模式而回到執行模式。如果MERLIN的系統磁碟還在磁碟機中，則將其

移出，並置入一片已設定過的工作磁片。

在'%'出現後，鍵入'S'（執行模式的儲存原始檔案命令）。系統會詢問你檔案名稱，鍵入'DEMO1' RETURN。在程式被儲存後，'%'又會出現。

2. 鍵入'C'（磁碟目錄），查看磁片的目錄。原始碼以"DEMO1.S"二進位檔案型式儲存在磁片上。附尾".S"是個檔案標籤，表示這個檔案是原始檔案。附尾詞是在'S'（儲存原始檔案）命令時，自動加在名稱的後面。

3. 鍵入RETURN回到執行模式，然後輸入'o'，表儲存目的碼。目的檔案是以先前指定原始檔案的名稱，儲存在磁片上，這並不會把原始檔案重寫過，因爲沒有詞尾會加在目的檔案名稱的後面。

當把檔案存入磁碟時，MERLIN也會同時顯示位址參數，並且計算及顯示長度參數。最好養成習慣將這些資料記錄下來。在查看磁碟目錄時，雖然執行模式所顯示的A$和L$參數，實際上並不作爲檔案名稱的一部分來存入磁碟中。如果你想在磁碟目錄上顯示這個訊息的話，可以用DOS RENAME命令。注意不要在新的檔案名稱內加上逗號。

鍵入'MON'， RETURN，於是監督程式的標記'*'出現。然後再鍵入'3ØØG' RETURN。你可以聽到'嗶'的一聲，表示這個範例程式工作了。

現在你可以鍵入CTRL-Y， RETURN，回到執行模式。

3.6 MERLIN 備份磁片的複製

要複製MERLIN的磁片，你必須利用在原來磁片中的複製程式。要執行這個程式，首先將系統磁片載入，然後從鍵盤上鍵入 'C'。這個程式會先查證你要複製磁片的意圖，然後再詢問 SLOT 及 DRIVE 的值。程式只能使用一部磁碟機來複製，如此才能提供最高的信賴度。

在複製的過程中，要小心適當的交換原來的和複製的磁碟片。如果在任何時刻，你犯了錯誤，立刻停止然後重新開始。

當複製結束後，建議你在日常的工作中使用備份磁片，並且把原來的磁片放在安全的地方。複製程式可以提供3個備份磁片的複製，所以一共有四個MERLIN組合器的磁片。此外，檔案可以從一片磁碟轉移到另一片上。這也就是說，損壞的檔案可以由另一片磁碟上確認完整的檔案來取代。利用你的Apple系統主磁片內的公用程式FID，把原來的主磁碟的檔案，轉移到損壞檔案的磁片上去。所有的檔案庫（ library file ），和磁片另一面。包括 SOURCEROR. FP 的程式，都可以利用FID公用程式把檔案移到任何一片 DOS 3.3 的磁片上。

4. 執行模式

執行模式是在程式階段，以提供檔案維護的操作，例如碼的載入、儲存，或是查看磁碟上的目錄。在下面各節中，將摘要說明執行模式(Executive mode) 中可下達的命令。

4.1 C: CATALOG (磁碟目錄)

在顯示完磁碟目錄後，你可以用標準DOS的語法，下達磁碟機命令。但不像LOAD, APPEND及SAVE等命令，當要參考到原始程式時，你必須鍵入詞尾“.S”。這種便利最主要是提供給檔案的鎖住(LOCK)和解鎖(UNLOCK)；不要利用它作載入及儲存檔案。如果你不需要下達磁碟機命令，只需鍵入RETURN 即可。若要取消鍵入一部分的命令，可用CTRL-X，或用後退鍵(Backspace)回到起始點。假如在“COMMAND:”後鍵入CTRL-C RETURN ，就會出現執行模式標號“%”。這時可以輸入執行命令，如L來載入檔案，而同時磁碟目錄還顯示在螢幕上。此外，當在“目錄暫停”時鍵入CTRL-C，會停止顯示下面的目錄。

4.2 L：LOAD（載入檔案）

這是從磁碟上載入一個原始檔案。此時MERLIN會要求你給予檔案的名稱。你不需要附加“.S”在檔案名稱之後，因爲MERLIN會自動做這件事。如果錯誤的鍵入“L”後，只要連續鍵入RETURN兩次，就會取消此次執行命令，並且不會影響在記憶體內的任何檔案。當輸入載入命令（或附加命令）後，系統會自動進入編輯模式，正如你輸入“編輯命令”一樣。此時，原始程式將被放在適當的位址上。隨後的“載入”或“儲存”命令會顯示上一次所用的檔案名稱，其後跟著一個問號“？”。如果你鍵入“Y”，就會以目前的檔案名稱執行命令；如果鍵入其他鍵（例如RETURN），游標就會移至第一個字元上，此時可以輸入想要的檔案名稱。如果只鍵入RETURN，就會取消此次命令。

4.3 S：SAVE（儲存檔案）

利用這個命令將原始檔案儲存在磁碟上，一如載入命令時，你不需要指定詞尾“.S”，而且連續鍵入RETURN，也會取消此次命令。注意：原始檔案的起始位址及長度會在主功能表（MENU）上顯示；但僅作參考用，你不需要利用這些參數。組合器會記憶這些參數並且自動送給DOS。和上一節載入檔案一樣，檔案名稱會顯示出來；如果鍵入“Y”，表示以相同的檔案名稱儲存在磁碟上。其他鍵則表示以新的檔案名稱儲存。

4.4 A : APPEND (附加檔案)

這項命令使得載入的檔案附加在目前存在記憶體的檔案後面，它和載入檔案的功能相同，而且不會影響到原來的檔案名稱。

4.5 D : DRIVE (磁碟機編號)

當鍵入" D "後，磁碟機編號就會從 1 變爲 2 ，或 2 變爲 1 。在主功能表會顯示目前使用的磁碟機編號。當最初 BRUN MERLIN 後，被選擇到磁碟機是編號 1 。MERLIN 並沒有命令來指定 SLOT 編號。但是可以利用下列步驟來達成。鍵入"C"查看目前的磁碟目錄，然後鍵入"CATALOG, Sn"，n 表 SLOT 編號。這個動作後，就會顯示新指定磁碟機上的目錄了。

4.6 E : EDITOR (編輯器)

這個命令會把系統置於編輯器／組合器模式下。它會自動設定編輯原始檔案的表格定位值。

4.7 O : SAVE OBJECT CODE (儲存目的碼)

此命令僅在原始檔案組合無誤後方可使用。在這種情形下，目的碼的起始位置及長度會顯示在主功能表，和原始檔

案的起始位置、長度一樣，這些參數僅供參考用。

注意：目的碼的起始位置是由程式的ORG值所設定（或假設值$8000），而並非實際記憶體的位置（$8000或者由OBJ所設定）。當執行此命令時，系統也會要求一個檔案名稱，但和原始檔案不同者，系統不會附加詞尾到檔案名稱後，所以你可以使用和原始檔案相同的名稱（當然！沒有“.S”）。當目的碼被儲存到磁碟上時，是以主功能表顯示的位址為準。以後，BLOAD或BRUN 該程式時，就會自動跳到該位址（由ORG所設定）。通常並不需要在原始碼加入OBJ，除非目的碼超過可用的空間（$8000以上）。

4.8 Q：QUIT（結束）

本命令使系統回到BASIC。可以輸入“ASSEM” 重新回到MERLIN系統，而不會破壞現在記憶體內的原始檔案。此命令可用來下達磁碟機命令，因為它比“C”（磁碟目錄）要來得方便。

4.9 R：READ（讀取文字檔案）

這是用來讀取MERLIN的文字檔（TEXT FILE）。檔案讀取後會附加在原來的緩衝區之後。當把緩衝區清除，重新開始後，讀取的檔案名稱就變成假設值。但是如果檔案讀取是附加在另一個檔案，檔案名稱的假設值就不會改變。

當讀取結束後，系統會進入編輯模式。如果檔案中某些行的長度超過255個字母：在讀取時，這些行會分成兩行或者更多行。除非記憶到了HIMEM， 否則檔案會一直讀取進來。當超過HIMEM時，系統會產生記憶錯誤，使得在HIMEM以前的資料才會存在。

系統在“讀取”和“寫入”文字檔時，會自動在檔案名稱前面加上“T.”。除非你在檔案名稱前加入空白(SPACE)或是其他介於$20和$40之間的ASCII 碼，否則都會被視為無效的輸入。

4.10 W：WRITE（寫入）

系統會將原始檔案以文字檔而非二進位檔寫入磁碟中。“讀取”和“寫入”的速度和“載入”及“儲存”的速度相當。在寫入完成後，系統會做查證(VERIFY)的工作。

5. 編　輯　器

基本上，編輯器中包含了三種模式：命令模式，附加或插入模式以及編輯模式。其中最主要的是命令模式，它的系統標號為冒號“：”。

5.1 命令模式

對於命令模式中的大多數命令而言，只有第一個字元是必須的，其餘都可供選擇。在手册中，必須的命令字元是以大寫字母出現，而選擇性的字元則以小寫字母出現。在某些命令，你必須指定某個列號，某段區域或一個區域表。列號只能是一個數字；某段區域是兩個列號組成，其間以逗點分開，一個區域表則包括了好幾個區域，其間以斜線（／）分隔。

某些命令允許字串的指定。字串必須以非數字的字元（不同於斜線）隔開。這種被分隔的字串叫 d 一字串（d-string），通常的分隔字元是使用引用符號（＇或”）。

在編輯器下，列號會自動產生。除非在你要下達命令時，否則絕不要鍵入列數在本文中。如果在一個區域中，列號

超過了最後一行，系統會自動調整到最後一行。以下是命令模式中的命令。

5.1.1 HImem

（在原始檔案與 38995 或者 $9853 之間的一個十進位或十六進位的數字）。這個命令很少使用。它會設定原始檔案的上限以及目的檔案的起始位址。除非你的目的位址不是用假設值，否則不需要去設定它。

5.1.2 NEW

清除現在的原始檔案，並且設定 HIMEM 爲 $8ØØØ。

5.1.3 PR#(0－7)

和在 BASIC 時的功能相同。主要用在列印原始檔案或是組合程式列表時，千萬不要用這個命令去選擇 80 行卡！

5.1.4 USER

系統會執行 JSR $3F5 指令。（相同於 Applesoft & 命令）通常在 $3F5 是擺上 RTS 指令。設計這個命令的目的在於連接使用者定義的列表機驅動器。（使用者的驅動器必須很小心地，避免用到第零頁的記憶位置，除非是 I/O 指標及 $60～$6F。要不然多半會干擾到 MERLIN 所使用的第零頁記憶位置）。

5.1.5 TABS

TABS 數字，數字…<定位字元>

這個命令會將編輯器的表格重新定位，但不會影響到組合程式的列表。表格最多可以定位到9個。系統假設的定位字元是空白鍵，但是可以由使用者指定任一字元。組合器把空白鍵視為唯一可接受的定位字元，來分隔標題（Labels），運算碼（Opcodes）和運算元（Operands）。如果你沒有指定定位字元的話，那麼上一個使用的定位字元被視為有效。輸入TABS後跟著一個CR將使得所有的表格定位設定為0。

5.1.6 LENgth

這個命令將顯示原始檔案的位元長度，以及在HIMEM（通常為$8ØØØ—— 而不是BASIC的HIMEM）之前所剩下的位元長度。

5.1.7 Where

Where（列號）

這將以十六進位方式印出該指定列號，實際在記憶體的起始位址。Where "Ø"（或"WØ"）會顯示原始檔案終止的位址。

5.1.8 MONitor

使系統回到監督程式控制下。重新回到MERLIN系統可以鍵入CTRL-C，CTRL-B或CTRL-Y。這些進入點會從MERLIN內部的保留區中取出保留值來，重新建立一些重要的零頁指標。就算是你在監督程式下，擾亂了一些零頁的指標，CTRL-Y還是會正確的進入MERLIN系統；但是此刻DOS並未連結上。這項功能是設計給熟練的Apple

程式師，至於生疏的使用者可以利用 ØG 直接進入編輯器。

不像其他的進入點，ØG 會使用零頁的指標 $ØA－$ØF，你必須確定這些位址沒有被改變過。

5.1.9 TRuncON

在 LIST 或 PRINT 時，這會設定一個旗號，把空白跟著一個分號（；）以後的資料截斷，而不顯示出來。因此，使得 Apple 40 行螢幕顯示原始檔案時，較易閱讀。然而在組合程式時，它會限制每列的目的碼最多爲 3 個；而不影響到註解欄。

5.1.10 TRuncOFf

這使得截斷旗號回復到假設值（在由執行模式進入編輯器，或者由組合器進入編輯器時，都會自動設定爲假設值。在組合器時，會導致列印所有的目的碼。

5.1.11 Quit

回到執行模式

5.1.12 ASM

把控制權交給組合器，把原始檔案加以組合。無論如何，首先會詢問你“是否要更改原始檔案？”，來提醒是否要更改資料或是原始檔案的辨證號碼。如果你回答爲“否”的話，就會開始程式組合。假如你回答“是”的話，那麼第一個包含有“／”符號的該列就會出現，並且將系統設在編輯模式下。當你編輯完這一列後，就會開始組合程式。假如你

利用 CTRL-C 編輯終結命令的話，那麼就會回到命令模式，並且你所建立的 I／O 開關都會被切斷；如 PR# 等。如果在原始檔案沒有一行包含著“／”符號，也會發生同樣的情形。

注意：藉著建立一個註解列“＊／”在程式的起始處，你就可以近乎自動地記著不同版本的程式。

5.1.13 Control-D

在程式組合的第二階段（pass 2）中，鍵入 CTRL-D 會反復列印旗號，使得列表可以停止或恢復。這好像在原始檔案中有一個 LST 運算碼，但是可經由 CTRL-D 來改變它。

5.1.14 Delete

Delete（列號）<區域><區域表>
Delete（區域）
Delete（區域表）

將使得指定的該列被刪除。由於列號不像 BASIC 是固定的，它會隨著插入及刪除而改變，因此你必須先指定最高的區域表。

5.1.15 Replace

Replace（列號）
Replace（區域）

這個命令會刪除指定的該列或區域，然後在該位置執行

插入模式。

5.1.16 List

List（列號）
List（區域）
List（區域表）

將原始檔案的列號及內容列印出來。控制字元除了送給列印機或其他非標準的輸出埠外，是以反相顯示。列表可以被CTRL-C或“／”鍵所終結。你也可以利用空白鍵來暫停列印，然後每敲一次空白鍵把下一列條列出來。其他的按鍵會使列印重新開始。在程式組合和列印符號表（SYMBOL TABLE）時，空白鍵的暫停功能依舊有效。

5.1.17 (句點)

從上次指定的區域起始處開始列印。舉例說明：如果你先鍵入L10, 100，於是第10列到第100列之間的資料，都會列印出來。然後你在利用“．”鍵，那麼列印就會從第10列開始，一直到結束（區域的最終位置並沒有被保留)。

5.1.18 / (列號)

這會使得從上一次列印列數的最後一列開始列印；或者從某個指定的列號開始列印。列印會持續到檔案的尾端或者被其他按鍵所停止（見5.1.16）

5.1.19 Print

Print （列號）
Print （區域）
Print （區域表）

除了列號不會被列印出來外，其餘和LIST命令相同。

5.1.20 PRinTeR （命令）

這個命令是將列印送到列表機，並包括了列印每頁的表頭（header）和自動跳頁的功能。利用配置程式可以改變假設的參數。它的語法如下：

PRTR slot# <字串> <每頁表頭>

如果 slot 編號超過 7 的話，JSR $3F5 的指令（&向量就會被執行，並且會連結在 $36－$37 位址的列印機驅動器上。

假如省略了每頁表頭，那麼每頁表頭就僅僅包含頁數而已。

起始設定字串不可被省略！如果列表機不需要特殊的字（串，則使用一個不會被承認的控制字元或是一個無用的字串在這種情形，可用 CR ）。例如 IDS列表機的起始設定字串為CTRL-Q；對大多數 Apple 的界面板而言。 CTRL-I 80N都可適用。

PRTRØ （不需要字串）會讓你在螢幕上看到每一頁的中斷點在那裏。假使在SLOT 3 有 80 行卡的，可以利用

PRTR 3 做此功能。除非 LIST, PRINT 或是 ASM 命令被確認後，輸出都不會送到列表機。

5.1.21 Find

Find（區域）
Find（區域表）
Find（d－字串）

此命令會將指定的字串統統列印出來。此命令也可利用 CTRL-C 或是“／”鍵來終結。由於在命令模式時，CTRL－L 會反復大小寫的開關，所以你也可以尋找或更換小寫字元。

5.1.22 Change

Change（區域）
Change（區域表）
Change（d－字串 d－字串）

更換是把第一個 d－字串更換爲第二個 d－字串。d 字串必須要具有相同的分隔字元，並且相鄰的分隔字元要合併在一起。舉例：要在 20, 100 區域間，更換“speling”爲“spelling，你就需要鍵入“C2Ø, 1ØØ speling spelling”。如果沒有指定區域的話，那麼整個原始檔案都會視爲更換區域。

在更換工作執行之前，你會被詢問是要“全部”（All）或是“部分”（SOME）。假如你回答是“部分”的話，鍵入“S”，編輯器會把含有第一個 d－字串的該列顯示出

來，然後停止下來等候更換。此時你若鍵入 ESC 或其他控制字元，就不會發生更換。其他的按鍵，例如空白鍵，就會產生更換。CTRL-C 以及 " /"鍵會終止更換的工作。

5.1.23 COPY

COPY（列號或區域）TO（列號）

把某一行或某一區域複製到指定的列號去。此命令並不會刪除什麼東西。

5.1.24 MOVE

MOVE（列號或區域）TO（列號）

和 COPY 相同，但在複製後，會自動刪除原來的區域。和移動前一樣，原始檔案會結束在同一行，但是次序卻不一樣了。

5.1.25 Edit

Edit（列號）
Edit（區域）
Edit（區域表）
Edit（d一字串）
Edit（列號）（d一字串）
Edit（區域）（d一字串）
Edit（區域表）（d一字串）

進入 EDIT 模式，並且在指定區域中，逐行地讓你編輯

。假如加上 d －字串，那麼只有那些包含 d －字串的原始列才會出現。

5.1.26　Hex-Dec Conversion

在命令模式時，如果你鍵入一個十進位的值（正或負），那麼相等的十六進位值會回覆於你。如果你鍵入的是十六進位（前置符號爲“ $ ”），那麼就會回覆十進位的相等值。所有的命令都接受十六進位值，所以在 HIMEM和 SYM命令時，是很方便的。

5.1.27　TEXT

將原始檔案中，所有的空白都轉變爲反相的空白。此命令的目的是讓你編輯一個“文字檔”（而非原始檔案），如此在列印檔案時，不需要每次都把表格定位歸零。除了會影響到編輯器的表格定位外，其他都沒有改變。

5.1.28　FIX

此命令解除“ TEXT”命令的效用。另外FIX也做了一些技術上的瑣事。建議你，當你從外部載入檔案，而在儲存該檔案之前，使用此命令，注意：TEXT和 FIX 都是以 SWEET-16 來寫的，通常會比較慢，對一個大的檔案，可能需要數分鐘。FIX或EDIT都會把每列中超過 255 長度的字元截掉。

5.1.29　SYM

MERLIN把符號表放在語言卡裏（ bank 1, $DØØØ —

$DFFF ）。除了非常大的程式外，4K的空間是相當適當的。萬一這空間用完後，可以藉著SYM命令組合器把符號表繼續放在另一塊區域去。例如，在你鍵入SYM$9ØØØ後，接著組合程式，如果在語言卡裏的4K空間用完後，就會從$9ØØØ開始擺上符號表，一直到達BASIC的HIMEM。要注意的是：HIMEM命令會取消SYM命令；或者你回到執行模式後，再進入編輯器時，也會取消此命令（在設定SYM前，先設定HIMEM）。設定SYM的位址必須介於HI-MEM以及BASIC HIMEM之間。如果符號表超過了配置的位址時，在組合程式的第一階段，會發出"記憶錯誤"的訊息。

5.1.30 VIDeo

此命令是用來選擇或是取消80行卡的輸出。假設值可以經由配置程式來選擇，這和在BASIC中的PR#的用法類似；但千萬別用PR#來選擇80行卡！PR#僅是設計給列表機使用的。假設80行卡的位置在slot 3，則可在編輯器下鍵入VIDEO 3來選擇80行卡。

若不再選擇80行卡的話，可以使用VIDEOØ或VIDEO $1Ø。這兩種型式都選擇了標準的Apple螢幕；但是VIDEO Ø會導致所有小寫字元變成大寫字元，呈現在螢幕上（但對列印機卻永遠不會改變）。如果你有小寫字元轉換器的話，那麼就可以使用VIDEO $10（或VIDEO 16)來取代VI-DEOØ。來作Apple螢幕的選擇。假使你的80行卡具有可以經由ESC程序控制，而回到40行模式的軟體開關的話，這些程序一樣可以使用。它的效應就像VIDEO $10一樣。

假如你沒有小寫字元轉換器的話，也可以在其後加上 VIDEO Ø。例如：對 Smarterm 的 80 行卡可用 ESC CTRL-Q RETURN；對 Sup"R"term 的 80 行卡，可用 ESC-Q-CTRL-X 等程序。

5.1.31 FW (尋字)

FW "字"

這是尋找（Find）命令的另一種型式。它可以找出那些僅僅爲非英文字母所包圍的指定字串。因此，FW"CAT"會找出：

```
CAT
CAT-1
(CAT,X)
```

至於 CATALOG 以及 SCAT，則不會被找出。

5.1.32 CW (換字)

CW "字" 新字

此命令工作的情形一如 FW。

5.1.33 EW (改字)

EW "字"

EW 相對於 EDIT，就好像 FW 相對於 FIND 一樣。

5.1.34 VAL

VAL "數學式"

組合器會計算該數學式的值出來，並回答給你。

範例：

VAL "LABAL"	在組合程式最後結束時，給予LABLE的位址（值）；或者如果找不到LAB-EL標題時，回答"未知標題"（unknown label）
VAL "$1ØØØ/2"	回答 $Ø8ØØ
VAL "%1ØØØ"	回答 $ØØØ8
VAL !"A"-"Ø"!	回答 $ØØ11

注意：對於命令中包含有字串的情形，" ^ "將被視爲"外卡"（"WILD CARD"）。因此F"Jon^s"會找出"Jones"和"Jonas"。

5.2 附加／插入模式

在編輯器中的附加／插入模式下，除了CTRL-R 不會有任何作用，以及離開附加模式的方法如下面所述外，其他功能與編輯模式相同。如：鍵入RETURN，會使編輯器接受在螢幕上顯示的整行資料。

5.2.1 Add

這將使你處在附加模式下，它的功能就如同在BASIC

中輸入一行的情形一樣，再加上自動設定列號的功能。然而你可以利用CTRL-L來輸入具有小寫字元的本文（如果你有一個小寫字元轉換器的話，對於程式的註解是非常有用的）。CTRL-L就像是個大小寫的反復開關，再一次的CTRL-L會使小寫狀態回復爲大寫狀態。若要離開附加模式，只要在該開頭鍵入RETURN即可。藉著鍵入空白以及RETURN，你也可以在檔案中輸入一個空列。因爲編輯器會自動地將每列尾端的空白去除，所以空列中的空白鍵並不會存在。你也可以鍵入CTRL-X或是CTRL-C來跳離附加模式；同樣地，也會取消現在該列的資料。

5.2.2 插入（Insert）

Insert（列號）

這會使你在指定的列號開始，輸入本文。至於其他的功能和附加模式一樣。

5.2.3 Control－L

反復現在的大小寫狀態。如果現在是大寫狀態，CTRL--L 會使得變爲小寫狀態，反之亦然。在輸入每一個新行時，假設值都是大寫狀態。

要改變一個字元組的大小寫狀態，只需鍵入CTRL-L，然後利用右箭頭（→）重打該字元組即可。

5.3 編輯模式

在編輯器中，鍵入E，就會進入編輯模式。在你指定區

域裏的第一列首先會出現在螢幕，而且游標在該列的第一個字元上。如同列印時，每列都會被表格定位，當以箭頭鍵（arrow key ）移動游標經過定位點游標會跳過此定位點到下個欄位去。當你完成編輯該行後，鍵入 RETURN 。

不管你在鍵入 RETURN時，游標的位置在那裏，顯示在螢幕上的該列會完全被接受。編輯命令以及編輯功能和 Neil Konzen's program以及 Southwestern Data System 的A.C.E.類似，但不完全相同。所有的編輯命令除了CT-RL-R外，都和附加／插入模式一樣。

5.4 編輯模式的命令

5.4.1 Control – I （插字）

開始插入字元，除了CTRL-L 外的控制字元會結束插字的功能，例如箭頭鍵及RETURN 。

5.4.2 Control – D （刪字）

刪除游標上的字元。

5.4.3 Control – F （尋字）

找尋接著CTRL-F 後鍵入的字元。這是有循環性的。

5.4.4 Control – O （插入特殊字元）

功能和CTRL-I 一樣，但是CTRL-O 可以插入任何一個控制字元（包括命令字元；例如CTRL-Q ）。

除了可插入控制字元外， CTRL-O 也允許使用者鍵入

那些在Apple鍵盤上無法得到的字元。CTRL-O接著鍵入:

<	gives	Control _
>	"	Control \
K	"	[
L	"	\
M	"	]
N	"	^
O	"	_
k	"	{
l	"	|
m	"	}
n	"	~
o	"	($FF ,無論任何機型)

注意:如果你有一個修改過的位移鍵,Shift-M可能會給予一個大寫的M(視你所有的是那一種),你必須利用CTRL-O才會得到一個右中括號"]"。

5.4.5 Control－P (重複＊＊＊)

如果在該列的第一個字元位置上鍵上CTRL-P時,就會讓該列重複32個星號。

5.4.6 Control－@ (首尾＊＊)

如果在該列的第一個字元上鍵入CTRL-@ 時,就會使該列的首尾爲星號,中間隔著30個空白。

5.4.7 Control－C或Control－X(取消)

結束編輯模式回到編輯器,而被編輯的該列會保持原來的形式。

5.4.8 Control－B（跳至列首）

將游標移動到該列的最前端。

5.4.9 Control－N（跳至列末）

將游標移動到列末的下一個位置去。

5.4.10 Control－R（回復）

回復該列成原來的格式。（在附加及插入模式下，沒有此功能。）

5.4.11 Control－Q（刪除游標後字元）

刪除游標後的字元。

5.4.12 Return（RETURN 鍵）

接受螢幕顯示的該列，並且編輯下一列，或者當編輯區域結束時，回到編輯器下。

編輯器會自動地把註解內的空白以及ASCII字串取代爲反相的空白。在列印時，再將它轉變回來，如此一來，你就可以不去理會它。其目的在避免錯誤地定位註解以及ASCII字串。

在ASCII字串的情況下，只有分隔字元是雙引號（"）或是單引號（'）時，才會這麼做。然而，你也可以完成同樣的工作，藉著編輯該列，用引號取代第一個分隔字元，鍵入RETURN，然後重新編輯，把分隔字元改爲原來的分隔字元。

6. 組　合　器

本章的說明，並不是要教你組合語言，而僅是解釋你所期望在原始檔案中要用到的語法，以及說明組合器所能提供給你的功能。

6.1　數值格式

組合器可以接受十進位、十六進位和二進位值的資料。十六進位必須前置一個錢號“$”，二進位則需前置百分比號“%”。下面四個指令是完全相等的。

```
LDA #1ØØ       LDA #$64       LDA #%11ØØ1ØØ       LDA #%Ø11ØØ1ØØ
```

由此例顯示，前置的零是可省略。# 字號在此處，表示“數值”或“資料”。四個指令的作用是把 1ØØ（十進位）這個數值放入累積器中。

沒有 # 號前置的數值，會視作一個位址。因此：

```
<LDA 1ØØØ       LDA $3E8       LDA %1111101ØØØ>
```

都是把記憶體位址 $3E8 裏的值放入累積器中。

利用適當的數值格式，使得程式變得清晰。例如，一個

資料表：

```
DA    $1
DA    $A
DA    $64
DA    $3E8
DA    $271Ø
```

就比相同十進位的資料表顯得神秘多了：

```
DA    1
DA    1Ø
DA    1ØØ
DA    1ØØØ
DA    1ØØØØ
```

6.2 原始碼格式

在一列中，典型原始碼的格式如下：

標題　　運算碼　　運算元　　；註解

如果一列中只有註解，那就必須在每列的起始處鍵入星號“ * ”；如果鍵入的是分號“ ; ”的話，那麼就會定位到註解欄去。組合器把原始碼中的一個空列當作跳行的指令（參見虛指令一節），而僅把列號列出來。

區分每一欄的空白數並不重要，除了組合器在列表外，每個欄位只需以一個空白隔開即可。

標題的最大長度可以到達 13 個字元，但是超過 8 個字元的標題會使得組合程式的列表顯得混亂。標題開頭字元的ASCII 值必須大於冒號，此外在標題的字元不得包含有小於零的ASCII 值。

組合器只檢查運算碼的前三位字元（對於 SWEET 16 的某些運算碼，如 POPD則例外）。例如，你可以用 PAGE 來代替 PAG（然而，第四個字元不可爲 D）。組合器在列表時會自動將運算碼截去成爲七個字元，除非沒有運算元，否則組合器對於運算碼超過四個字元以後的字元皆不會檢查。

運算元加上註解欄的最大容許長度爲 64 個字元。如果超過了這個限制，你會得到錯誤訊息。而一個註解行的最大長度也是限制在 64 個字元。

6.3 數學式

爲了要讓組合器接受以及(或者)需要清晰的語法，我們必須定義所謂的"數學式"。數學式是由"基本式"的算術和邏輯運算所組成的。所謂"基本式"（ primitive expression ）是指：

1. 一個標題
2. 一個十進位值
3. 一個十六進位值（詞首爲" $ "）
4. 一個二進位值（詞首爲" % "）
5. 任何一個ASCII值，當它的詞首爲單引號或雙引號；或被其包圍時。
6. 星號（ * ）（表示現在的位址）

所有的數值系統都是 16 一位元的資料，前置零是不需要輸入的。在第 5 種情況時，基本式的"值"，就是該字元的 ASCII值。如果使用雙引號的話，那麼最高位元會設定爲

“ 1 ”；反之使用單引號的話，最高位元會清除爲“ 0 ”。

組合器提供了四種算術運算：+，－，／和＊；以及三種邏輯運算：！表示 Exclusive OR（互斥）；．表示 OR（或）；和 & 表示 AND（及）。

下面範例表示合法的數學式：

```
LABEL1-LABEL2
2*LABEL+$231
1234+%10111
"K"-"A"+1
"0"!LABEL
LABEL&$7F
*-2
LABEL.%10000000
```

括號具有別的意義，不可在數學式中使用。所有算術及邏輯運算，都是由左至右（2 + 3 ＊ 5 會得出 25 而非 17）。

6.4 立即資料

某些運算碼，如 LDA，CMP 等等，皆可以接受立即資料（數值，不同於位址）。立即資料的型式是在數學式的前面加上“ # ”號。例如 LDX #3。此外：

# ＜數學式	產生數學式的低位元組
# ＞數學式	產生數學式的高位元組
# 數學式	同樣是低位元組（6502 並不接受兩個位元組的資料）
# ／數學式	產生數學式高位元組的另一種語法

具有計算數學式（例如 LAB1—LAB2－1）能力的組合

器，對於下列型式的原始碼是非常有用的：

```
COMPARE        LDX     #FOUND-DATA-1
LOOP           CMP     DATA,X
               BEQ     FOUND
               DEX
               BPL     LOOP
               JMP     REJECT    ;not found
DATA           HEX     E3BC3498
FOUND          RTS
```

這種型式的原始碼，當你增加或刪除某些“資料”時，那麼在比較迴路間的X索引值，會自動調整到適當值。

6.5 定址模式(6502運算碼)

所有標準的6502助憶符號，都可爲組合器接受。此外還可接受 BLT（小於時分歧）—相當於 BCC；以及 BGE（大於等於時分歧）—相當於 BCS。

6502具有12種定址模式。這12種模式的語法，MERLIN是這樣規定的：

範　例：

語法

```
Syntax
Implied                OPCODE              CLC
Accumulator            OPCODE              ROL
Immediate (data)       OPCODE #expr        ADC #$F8
                                           CMP #"M"
                                           LDX #>LABEL1-LABEL2-1
Zero page (address)    OPCODE expr         ROL 6
      Indexed X        OPCODE expr,X       LDA $EØ,X
      Indexed Y        OPCODE expr,Y       STX LAB,Y
Absolute (address)     OPCODE expr         BIT $3ØØ
      Indexed X        OPCODE expr,X       STA $4ØØØ,X
      Indexed Y        OPCODE expr,y       SBC LABEL-1,Y
Indirect               JMP    (expr)       JMP ($3F2)
      Preindexed X     OPCODE (expr,X)     LDA (6,X)
      Postindexed Y    OPCODE (expr),Y     STA ($FE),Y
```

注意：零頁和絕對定址模式的語法並無不同。組合器會適當地選用零頁。在間接索引定址的模式，數學式必須是零頁，否則組合器會發出一個“數學式”不是零頁的錯誤訊息出來。

注意：“累積器”模式並不需要運算元。有些組合器會要求你輸入一個“A”做爲運算元。

組合器會判定給予運算碼的定址模式是否合法。

此外，MERLIN還提供了強迫非零頁的定址，只要在運算碼的後面加上任何字元（除了“D”以外）。例如

```
LDA  $10     組合爲零頁（2個位元組）
LDA: $10     組合爲非零頁（3個位元組）
```

6.6 SWEET 16運算碼

組合器可以接受所有標準的SWEET 16運算碼。常用SWEET 16的16個暫存器，RØ到R15，不需要相等於某個值，“R”是可以選擇的。TEDILII+的使用者將會很高興地看到SET運算碼正常的工作（跟著數值或標題）。對SET運算碼而言，運算元的暫存器和資料部位之間，可以用逗號也可以用空白分開；也就是說，SET R3, LABEL 和SET R3 LABEL是相同的。要注意的是：NUL運算碼會被組合爲一個位元組的運算碼（和HEXØD相同）和ROM SWEET 16所組合成的跳過兩個位元組有所不同，基於內部的原因，這是故意如此地。

6.7 虛運算碼一指令

6.7.1 EQU(=)

EQU 數學式(EQUals)
= 數學式(另一種語法)

用做定義LABEL的值，通常用來作爲一個外部的位址，或是對於常用的常數，而希望給予一個有意義的名稱。建議所有EQU值放在程式的開始。如果某標題用到一個對等值，而該對等值的定義又放在該標題的後面時，組合器就不會將其組合爲零頁。因此，可能會收到“錯誤碼”的訊息。(亦參見“ Variables ”)

6.7.2 ORG

ORG 數學式(ORiGin)

設定程式開始執行的位址。假設(default)值爲HI-MEM的假設值爲$8ØØØ)。正常下，只要一個ORG值，而且放在程式的開始處。對於含有多個ORG的程式，第一個ORG會給予BLOAD的位址。這對產生一個目的檔案，使它可以從一個位址載入，也可以經由ORG設成另一個位址執行。

你不能使用ORG*−1 等，來使目的指標後退一位。雖然有些組合器可以這麼做，但是在MERLIN，你必須用DS−1來取代這種功能。

6.7.3 OBJ

OBJ 數學式（ OBJect ）

設定在程式組合時，設定目的碼的位址。通常假設值在 HIMEM。這個虛運算碼甚少使用，對於一個無經驗的程式師，最好不要去用它。如果 OBJ 設定在 BASIC HIMEM（或者 SYM位址）以上時，將不會產生目的碼。這可以用來把一大段的列表送給列印機，或者在程式組合時，直接把目的碼送給磁碟機（ DSK 運算碼）。

6.7.4 PUT

PUT 檔案名稱

PUT 檔案名稱（可接受標準DOS 語法中的 SLOT 及 DRIVE 參數）會使被命名的檔案被讀取進來，然後"插入"在運算碼的位置。（除非該檔案名稱第一個字元小於"@"，否則會附加詞首"T"到被命名的檔案名稱前面）。

注意："插入"是對一序列原始檔案而言，而其實際上是插入在主檔案記憶位置的後面。爲了保護記憶體，只有文字檔具有這個功能。如果被 PUT 的檔案長度超過 HIMEM，會導致記憶錯誤的訊息。一次只有一個檔案會被插入到記憶體內，所以可以利用這個功能，把很大的程式加以組合。

使用 PUT 有兩個限制：第一，在 PUT的檔案不可以有集體指令（macro）的定義。集體指令的定義必須放在主原始檔案裏。第二，被 PUT 的檔案，不能以 PUT 虛指令去呼叫

另一個 PUT 的檔案。當然在主程式中，可以包含集體指令的定義以及藉著 PUT 運算碼來輪流呼叫其他檔案，藉以，來模擬檔案的鏈結。

任何的變數（如]LABEL）被視爲“本地”（local）變數而可以被使用。常用的本地變數從]1 到] 8 可以藉著 VAR 運算碼而建立。

PUT 的功能提供你一個簡單的途徑，來把一些常用的副程式，例如 MSGOUT 或 PRDEC，納入你的程式之中。

6.7.5 VAR

VAR expr;expr;expr …

這是一個很方便的途徑，使得變數] 1 到] 8 去對等一些值。“VAR 3;$42;LABEL”會使得] 1=3,] 2=$42 以及]3=LABEL。這個設計是專爲在 PUT 之前使用的。如果一個 PUT 的檔案使用到]1 到]8，除非是在呼叫集體指令時使用＞＞＞；否則，必須預先宣告這些變數。

6.7.6 SAV

SAV 檔案名稱

SAV 檔案名稱（可以接受 slot 和 drive 參數）將把現在的目的碼以指定的名稱儲存在磁碟上，就好像執行模式下的儲存指令一樣，但是它可以在程式組合時，重覆做好幾次。

當程式有兩個以上 ORG 時，這個虛運算碼可以提供儲存一部分目的碼的方法。因此可以用來組合一個非常大的程式。在儲存後，目的位址會重新設定爲上一次所指定的 OBJ

或假設的HIMEM值。

儲存命令會把被儲存檔案的起始位置設定到正確的位址。例如，你的程式如果包含有3個SAV命令，就會把程式分爲三段來儲存。當稍後BLOAD時，三個程式會一個接一個地載入到正確的位址上。

把PUT和SAV操作碼聯合使用，可以把非常大的檔案加以組合。

6.7.7 DSK

DSK檔案名稱

DSK檔案名稱這虛運算碼，會指引組合器把接著的碼組合到磁碟機去。當DSK在操作時，會關閉舊的檔案而開始一個新的檔案。這對於很大檔案的處理是很有用的，但對於中等大小的檔案，使用SAV是比較好的，因爲SAV比DSK要快30%，而且在理論上也更值得信賴，因爲在DSK操作時，CHK操作碼和DSK並不配合，而會被取消掉。

6.7.8 END

通常較少使用，或者是需要通知組合器不要組合這個虛指令以後的原始檔案。在END後面出現標題是不會被承認的。

6.8 定格式

6.8.1 LST ON/OFF

LST ON或OFF (LIST)

這個指令控制組合器，把程式組合的列表送給螢幕以及（或者）其他輸出裝置。舉例說明，你可以把程式組合的一部份輸出給列表機。在原始檔案中，使用LST的次數並不受限制。如果在程式組合完畢之前，LST的狀態是OFF的話，那麼符號表就不會印出。實際上，組合器是檢查運算元的第三個字元是否爲一個空白。因此，LST ERINE 和 LST OFF的效果相同。LST指令並不會產生目的碼。如果LST的狀態爲OFF的話，目的碼的產生會快得多，但是這個情況，只建議你使用在已經除錯的程式上。

注意：在程式組合第二階段時，由鍵盤輸入 Ctrl -D 會反復LST的狀態。

6.8.2 EXP ON/OFF

EXP ON或OFF（EXPand）

EXP ON 會在程式組合時，把整個集體指令的內容印出。而在OFF狀態時，只印出PMC虛運算碼。EXP是假設爲ON。這對於目的碼的產生並無任何影響。

6.8.3 PAU

PAU(PAUse)

在程式組合第二階段時，會使得組合的程序暫停直到有按鍵輸入爲止。這和在鍵盤鍵入空白的功能一樣。

6.8.4 PAG

PAG (PAGe)

這會使得控制字元 $8C (換頁) 輸出給列表機。 但對螢幕的輸出，即使是使用 80 行卡也不會有影響。

6.8.5 AST

AST數學式 (ASTerisks 星號)

會送出和運算元相同值的星號來。所採用的數值格式爲最普通的，例如：AST 1Ø 會送出 10 個星號。數值是以 256爲模數，所以 Ø 表示 256 個星號。這和 TED II+ 有所不同，後者需要運算元爲十六進位的數學式，而且需要經過轉換。

6.8.6 SKP

SKP 數學式 (跳行)

在列印時，送出和運算元相等的 carriage return(印字頭回轉， OD) 來。數值的格式和 AST 相同。

6.8.7 TR

TR 運算碼

和在編輯器的TR命令一樣。TR 或是 TR ON 使得列印目的碼時，每列最多只有三個位元組。TR OFF使得以列印所有的位元組。

6.9 字 串

6.9.1 ASC

ASC d－字串（ASCII）

將會使一個分隔ASCII 字串放到目的碼中。分隔字元的唯一限制是它不能在字串本身中出現。不同分隔字元具有不同的效應。任何一個分隔字元，它的ASCII值小於單引號時會使得最高位元為ON；反之，則最高位元為OFF。例如：分隔字元！“＃＄％＆ 等會產生一個“負”ASCII的字串；而′（）＋？會產生“正”ASCII。通常單引號（′）和雙引號（″）是最常被選擇的。但是其他的分隔字元可用來分隔，包含單引號或雙引號的字串。

6.9.2 DCI

DCI d－字串（Dextral Character Inverted右側字元反相）

這和ASC 是一樣的，但是字串的最後一個字元的最高位元和前面的相反。

6.9.3 INV

INV d 一字串（ INVerse 反相）

使得放在記憶體的字串爲反相的格式。所有分隔字元都有同樣的效應。

6.9.4 FLS

FLS d 一字串（ FLaSh 閃爍）

使得放在記憶體的字串爲閃爍格式。所有分隔字元，都有同樣的效應。

6.9.5 REV

REV d 一字串（ REVerse 反向）

例如： REV "DISK VOLUME" 會導致

EMULOV KSID（分隔字元的選擇一如ASC）

6.10 資料和空間配置

6.10.1 DA

DA 數學式（ Define Address 定義位址）

把運算元的值（兩個位元）儲存在目的碼，低位元組在前面。DA 通常用來定義位址。例 DA $FDFØ 會產生 FØ FD 的目的碼。在一列中可接受多項的資料（例如： DA

1, 10, 100)

6.10.2 DDB

DDB 數學式 (Define Double-Byte 定義兩個位元組)

和前述一樣，但是高位元組會放在前面；也可在一列中接受多項資料 (例如：DDB 1, 1Ø, 1ØØ) 。

6.10.3 DFB

DFB 數學式 (DeFine Byte 定義位元組)

把指定的運算元以一個位元組的長度放在目的碼中。DFB 可以接受幾種型式的資料，但是資料間必須以逗號分隔，而且不可有空白存在，通常都使用標準的數值格式和算術運算。" # " 字號是可接受的，但和 " < " (小於) 符號一樣被忽略不用。" > " (大於) 符號表示是指定標題的高位元組，若無此符號，則表示低位元組。" > " 必須出現在數學式的第一個字元，或者恰好在 " # " 之後，也就是說，指令 DFB >LAB1-LAB2 會產生 LAB1-LAB2 值的高位元組。

範例：

```
DFB $34,1ØØ,LAB1-LAB2,%1Ø11,>LAB1-LAB2
```

對於 DFB 的陳述式而言，上面的指令是正確的。假設 LAB1=$81A2 和 LAB2=$77C4.，那麼會產生 34 64 DE ØB Ø9 的目的碼 (十六進位) 。

6.10.4 HEX

HEX十六進位資料

這是DFB的另一種型式，對於十六進位的資料給予了很方便的插入。不像其他的指令，“$”符號並不需要而且也不為其所接受。運算元必須由十六進位的數值所組成，而每個數值必須有兩個十六進位的數字，例如ØF而非F）數值資料間可以相鄰，也可以用逗號隔開。下面的情形將會導致錯誤。運算元包含了奇位數的數字；或者以逗號出現在行尾；或者（對所有的情況）每行超過64個字元。

6.10.5 DS

DS數學式（Define Storage定義記憶）

為了字串的記憶資料而預留空間。DS並不產生目的碼。例如DS 1Ø會預留1Ø位元組的記憶空間。由DS會調整目的碼的指標，所以像DS－1的指令，可以用來把目的碼指標和位址指標後退一個位元組。

6.10.6 KBD

KBD（KeyBoarD鍵盤）

在程式組合時，允許一個標題由鍵盤輸入。它的語法是：LABEL KBD。

6.10.7 LUP

LUP數學式（LOOP迴路）

－－∧（迴路結束）

迴路指令的範例如：

```
LUP 4
ASL
--^
```

這將被組合成：

```
ASL
ASL
ASL
ASL
```

程式在組合列表時，也會列印成上述情形，但列號是重覆的。

也許LUP最主要的用途是在建表格。例如：

```
]A      =    Ø
        LUP  $FF
]A      =    ]A+1
        DFB  ]A
        --^
```

上例會產生表格 1, 2, 3, … , $FF。迴路最大的值是 $8ØØØ，如果廻路值超過此數，LUP運算碼就會失效。

6.10.8 CHK

CHK 數學式（ CHecKsum 檢驗和 ）

這會使得目的碼在CHK運算碼的位置上放進去一個位元組的檢驗和（通常是在程式的後面）。當使用DSK運算碼，不能使用此運算碼。

8.10.9 ERR

ERR 數學式（ERRor 錯誤）

如果數學式的值不爲零的話，會強迫產生錯誤，並送出“在？？？列中斷”的錯誤訊息。

在程式最後一列加上此指令，可以用來確保你的程式不會過大而超過界限（例如 $95FF）

```
ERR   *-1/$9600
```

注意：這僅是來防止你在程式組合時，由於程式太大，而寫入資料到 $9600 以後，但對程式本身並無損害。錯誤只會在程式組合第二階段時發生，但錯誤並不會結束程式組合。另外一個用得到的語法是：

```
ERR   ($300)-$4C
```

如果在 $300 位置並沒有擺上 $4C 的話，程式會在第一階段時發出錯誤，並且結束程式組合。

8.10.10 USR

USR（運算碼）

這是一個使用者定義的虛運算碼。實際上，組合器會執行 JSR $B6DA。在開機載入時，或者 BRUN MERLIN 和 BRUN BOOT ASM 時，會在此位址上塡入 RTS 。爲了要建立你自己的程序，你必須在執行模式的 CATALOG 命

令後去BRUN它。這僅是在$B6DA上擺上JMP 指令到主常式（ routine ），然後RTS 。下列旗號以及進入點（ entry point）可能為你的常式所使用：

```
USRADS   = $B6DA    ；必須是JMP到你的常式
PUTBYTE  = $E5F6    ；參見下面
EVAL     = $E5F9    ；參見下面
PASSNUM  = $2       ；包含組合階段數
ERRCNT   = $1D      ；錯誤計數
VALUE    = $55      ；由EVAL返回的值
OPNDLEN  = $B3      ；包括運算元和註解的聯合長度
NOTFOUN  = $FD      ；參見EVAL
WORKSP   = $280     ；包括正ASCII的運算元及註解
```

你的常式會被USR運算碼以A＝∅，Y＝∅和設定溢位旗號所呼叫，要讓組合器在目的碼中擺上一個位元組，你應該JSR PUTBYTE，並以該位元組的值在A暫存器中。

PUT BYTE會保留Y的值，但會破壞A和X。它返回時，零旗號恒為清除（所以BNE會分歧）。在第一次階段時，PUTBYTE調整目的和位址指標，使得暫存器裏的值並不重要。你必須在每一個階段中，呼叫PUTBYTE 相同次數；否則指標將不會保持正確，而且在組合程式的其他部分時將會發生錯誤。

如果你的常式需要計算運算元或者一部分的運算元，你可以JSR EVAL來達成。X暫存器必須指向你要計算的運算元該部分開始的第一個字元（X＝∅是從運算元的數學式起始處計算起）。由EVAL返回時，X會指向數學式的下一個字元。Y暫存器會是∅，1或2相對應於字元右大括號，空白以及逗號。

任何不允許的字元出現在數學式中時，會導致程式組合結束，並回復“ BAD OPERAND”(錯誤運算碼)。如果數學式中的某些標題不被承認時，那麼在NOTFOUND 位址的值會不為零。在第二階段時，你會收到“ UNKNOWN LABEL” (未知標題)的錯誤，並且使你餘下的常式被忽視。在從EVAL返回時，數學式的計算值會存在VALUE和VALUE＋1 位址裏。(低位元先)。在第一階段時，如果NOTFOUND 值不為零的話，這個值並不重要。

你的常式適當的位址是在 $300 ～ $3CF 以及 $8A0－$8FF。但絕對不能寫到 $900 。對於較長的程序，你可以使用較高位址的記憶體，但要在 $ 9853 以下 。如果你確定你的符號表不會超過 $1ØØØ 位元組的話，你可以利用SYM的編輯命令來保護你的常式不會被目的碼重覆寫入。SYM要設在你的常式的下一個位元組 以後。你可以使用零頁 $60－$6F ，但不能改變其他的位址，再者 $226－$27F 以及 $2C4－$2FF 裏的內容不可被更改。在從你的常式返回時(RTS)，USR這一列將會被印出(在第二階段)。

若要進一步了解USR的用法，可閱讀原始檔案 SCRAMBLE. S，或者更複雜的程式範例，FLOAT. S 。第一個檔案使用USR運算碼將ASCII 字串放進一個混雜格式的目的碼。第二個檔案則是較複雜的程式，它使用Applesoft來計算一組(五個位元組)指定的浮點數值(Floating point number)，而且把它投入目的碼裏。在這裏，後者僅能在Apple II plus 的機型上才能組合。

當你在原始檔案中使用USR運算碼時，最好包含某些

記憶體中所需常式的檢查。例如：你的常式在位址＄31∅中包含有＄31的值，然後

```
ERR ($31∅)-$3
```

就會測試那個位址的值是否正確，如果不正確，就會結束程式組合。相同地，如果你知道你所需的常式應該組合成兩個位元組的資料，那麼你可以用下列的碼來做（粗略的）檢查：

```
LABEL      USR  OPERAND
           ERR  *-LABEL-2
```

如果USR操作碼沒有產生兩個位元組的目的碼，就會在第二階段強迫發生錯誤。

在同樣的原始程式裏，可以使用USR在好幾個不同的常式中。例如：你的常式可以檢查第一個運算元的數學式，做爲所要常式的索引，然後據以產生各種行動。例如"USR 1, whatever"會分歧到第一個常式，"USR 2, stuff"分歧到第二個常式等等。

6.11 條件組合

6.11.1 DO

DO數學式

這個運算碼和ELSE及FIN一起使用，就構成了條件組合的虛運算碼。如果運算元的值爲零的話，組合器就會停止產生目的碼（直到下一個條件出現）。除了集體指令的名稱

，組合器不會承認在此區域的任何標題。如果運算元的值不爲零的話，那麼組合器就會照常進行。這對於集體指令是非常有用的。對於不同情況要產生稍微不同目的碼的常式，也是很有用的。例如：如果你設計一個常式能在ROM裏進行，你可能希望有一個ROM的版本；爲了除錯方便，你可以產生一個稍微不同RAM版本。

同樣的，在一個本文程式中，你也可能希望一種版本適合於有小寫字母轉換的Apple，以及另一種版本適合於沒有小寫字轉換的Apple。藉著使用條件組合，使得修改這種的程式非常簡單；因爲你不需要修改兩個不同版本的原始碼。每一個DO指令，必須由FIN指令來作結束；而每一個FIN指令，必須有DO指令在其之前。ELSE指令必須在DO和FIN之間。DO，FIN這種結構，最多可以有8個巢狀組織（其中可能包含ELSE）。如果DO的條件爲OFF（值爲零），那麼除了相對的FIN出現時，或者，ELSE在此出現，否則程式組合不會回復作用。巢狀的DO，FIN結構，在產生條件式的集體指令是很有用的。

6.11.2 ELSE

對於上一個DO指令，這會使得組合的條件相反（ON-->OFF或者OFF-->ON）。

6.11.3 FIN

取消上一個DO指令。

6.12 集體指令

6.12.1 MAC（MACro 集體指令）

這是一個集體指令定義開始的信號。它必須給予一個集體指令名稱。接著，你所使用的名稱就會被保留起來，而且除了是 PMC 虛運算碼外，其他的程式不能用到這個名稱（例如 NAME 爲集體指令的名稱，那 DA NAME 就不爲組合器接受）。然而可以在集體指令前面擺 LABEL EQU * 或者 LABEL DS Ø 來模擬。通常這是極少需要這麼做。集體指令的詳細用法，請參閱第七章。

EOM (<<<)

EOM (End Of Macro(集體指令結束)
<<< (另一種語法)

這表示集體指令定義的結束，它可以擁有標題以用來分支到集體指令的結束。

6.12.2 PMC（>>>）

PMC 集體指令名稱（Put MaCro）
>>>集體指令名稱（另一種語法）

指示組合器在現在的位址上，複製一份命名的集體指令。參閱第七章。PMC 可以擁有標題。

6.13 變 數

凡是起首爲“] ”的標題，都視爲變數。這些變數可以由EQU來定義，而不能用來標誌其他的事。你可以視需要重新定義它。變數設計的目的是給集體指令中使用的，但是並不侷限於此。

向前參考 (Forward reference) 一個變數是不可能的（以正確的結果），但是組合器會指定某些值給它。也就是說，必須在使用前定義一個變數。

7. 集體指令

7.1 定義集體指令

集體指令的定義開始爲：

NAME MAC（不須運作元）

集中NAME爲標題，集體指令用EOM或<<< 來結束。而NAME這個標題只能用PMC NAME或>>>NAME來呼叫。

只須在程式中第一次用到此才定義此集體指令。不過，最好（當集體指令用到變數時）在程式一開始就定義所有的集體指令，且在"條件組合"爲OFF（CONDITIONAL ASSEMBLEY OFF），然後需要時呼用集體指令。

不可向前呼叫集體指令（也就是集體指令定義較晚），不然會有NOT MACRO（非集體指令）錯誤出現。也就是說要先定義集體指令然後才可呼叫。

集體指令的定義裏面可以使用"條件組合"，如DO, ELSE, FIN。

集體指令定義裏的標題在每次呼叫集體指令時都會自動的調整。請參考7.4節例題中的LOOP與OUT。

集體指令中所產生的錯誤常常使得組合中斷，因爲其所產生的錯誤都是非常具破壞性的。錯誤訊息所指出的列號是呼叫集體指令那列的列號，而不是集體指令裏面的列號。

7.2 巢狀式的集體指令

集體指令中可以再呼叫以前定義過的集體指令，最多可以有 15 層。當在集體指令中呼叫其他集體指令時，必須在 DO 條件爲 OFF 之下方可。

以下是一個例子：

```
TRDB MAC
     >>> TR.]1+1;]2+1
TR   MAC
     LDA ]1
     STA ]2
     <<<
```

上面的例子中>>> TR. LOC;DEST 將會被組合爲:

```
LDA LOC
STA DEST
```

而>>>TRDB.LOC;DEST 將會被組合爲：

```
LDA LOC+1
STA DEST+1
LDA LOC
STA DEST
```

另外舉一個較通常的例子。(其中 CH=$24)：

```
POKE MAC
     LDA #]2
     STA ]1
     <<<

HTAB MAC
     >>> POKE.CH;]1
```

7.3 特殊的變數

有 8 個特別的變數（名稱爲]1 到]8）已先行定義了，以方便在集體指令中使用。這些是用在集體指令的呼叫指令中：

```
>>> NAME exprl;expr2;expr3...
```

就設定 expr 1 爲]1， expr 2 爲]2，餘類推。舉例如下:

```
TEMP    EQU     $10
        DO      0
SWAP    MAC
        LDA     ]1
        STA     ]3
        LDA     ]2
        STA     ]1
        LDA     ]3
        STA     ]2
        <<<
        FIN
        >>>     SWAP $6;$7;TEMP
        >>>     SWAP $1000;$6;TEMP
```

這段程式把 $6， $7 位址裏的值互換，用 TEMP 做爲暫時的存放位址，接著把 $6 和 $1ØØØ 位址裏的值互換。

如上顯示，呼叫集體指令時必須設定 expr 1 等，以相對應其定義裏的變數。在程式組合設定時會把相對應的變數換爲 exprl， expr2 等。變數的數目必須與 expr 的數目相等，若不相同就會產生" BAD OPERAND"（錯誤運算元）的錯誤信息。但若 expr 的數目超過變數不會有錯誤信息產生。

集體指令與 expr 中間可用的符號包括（除了空白字元外）：

. / , - (

通常都用“ . ” 。expr 與 expr 間必須用“ ; ”分開且不許有額外的空白。

集體指令可接受多種型式的資料，所以組合器可以接受下列的集體指令呼叫：

```
        DO   0
MUV     MAC
        LDA  ]1
        STA  ]2
        <<<
        FIN
        >>>  MUV.(PNTR),Y;DEST
        >>>  MUV.#3;FLAG,X
```

或者：

```
        DO   0
PRINT   MAC
        JSR  SENDMSG
        ASC  ]1
        BRK
        <<<
        FIN
        >>>  PRINT.!"quote"!
        >>>  PRINT.'This is an example'
        >>>  PRINT."So's this, understand?"
```

限制：如果上述的字串包含空白或者分號，則必須用引號（單引號或雙引號）來做分隔號。再者，如>>> WH-AT.“A”其最後的雙引號也是必須的。（這雖只在集體指令呼叫或VAR指令中受限，但最好是所有指令都如此用）。

MERLIN早期的版本是用逗點而不是分號來分隔expr,那些早期版本的使用者，可以用磁片所附的程式“CONVERT ”來轉換；先把原始程式載入記憶體，在CATALOG之後，於執行模式時BRUN CONVERT。

7.4 程式範例

以下的程式說明非標準變數的用法，但如用標準變數]1代替]MSG可使程式看起來悅目些（若用]1代替]MSG則可省去]MSG EQU HITMSG這行，而在>>>… 行中設定]1）：

```
HOME      EQU    $FC58
COUT      EQU    $FDED
KEY       EQU    $C000
STROBE    EQU    $C010
DOS       EQU    $3D3

          DO     0           ; 不組合
SENDMSG   MAC                ; 開始定義 SENDMSG
                             ; 集體指令
          LDY    #0
LOOP      LDA    ]MSG,Y      ; 取一個字母
          BEQ    OUT         ; 結束輸出
          JSR    COUT        ; 輸出
          INY
          BNE    LOOP        ; 回去再送

OUT       <<<                ; 結束集體指令定義，且結束此
                             ; 程式
          FIN
          JSR    HOME
]MSG      EQU    HITMSG
          >>>    SENDMSG
INVRS     CMP    #"I"
          BNE    NORM
]MSG      EQU    IMSG
          >>>    SENDMSG
```

```
NORM     CMP    #"N"
         BNE    STP
]MSG     EQU    NMSG
         >>>    SENDMSG
STP      CMP    #"S"          ; 是否結束
         BNE    GETKEY        ; 不是，取下個輸入
         JMP    DOS           ; 整個結束，回到DOS

HITMSG   ASC    !HIT
         HEX    8D8D00
FMSG     FLS    "THIS IS A FLASHING MESSAGE"
         HEX    8D8D00
IMSG     INV    "THIS IS A MESSAGE IN INVERSE"
         HEX    D8D00
NMSG     ASC    "THIS IS A NORMAL MESSAGE"
         HEX    8D8D00
```

7.5 集體指令程式庫

本磁片附有一集體指令程式庫，包括三個範例程式，此程式庫的目的是給對集體指令不熟者一個入門，提供示範如何在程式中使用集體指令。

注意：原始程式一開始即行定義所有用到的集體指令，而每個範例程式在須用時才呼叫集體指令，並用"條件組合"以決定要組合那一個程式，KBD這個運算碼可以讓使用者在執行組合程式時決定要組合那一個程式。

8. 技術資料

原始程式由 $9Ø1 放起，不管程式 ORG 的位址。

有三個重要指標：

程式起始位址指標：$A, $B （一定設定為 $9Ø1）
程式高限位址指標：$C, $D （假設為$8ØØØ）
程式結束位址指標：$E, $F

當你進入監督程式或 BASIC 時這些指標已存在 $EØØA 到 $EØØF，當再進入MERLIN時會再重新填入零頁 $A 到 $F。

當進入MERLIN時會把 I/O 開關再換回標準的 I/O 開關。並且DOS再度連接上，有如從鍵盤鍵入 PR#Ø 及 IN #Ø。進入編輯模式時，DOS就從 I/O 開關上移走，這樣就可以使用INIT做標題而不致於因少許錯誤引發不可收拾的災難。在編輯模式下以PR#命令所建立的任何 I/O 開關，一進入執行模式時即自動斷開，並且連上DOS的開關。組合程式執行完後或被CTRL-C中斷後DOS的 I/O開關且會被斷開。

8.1 一般技術資料

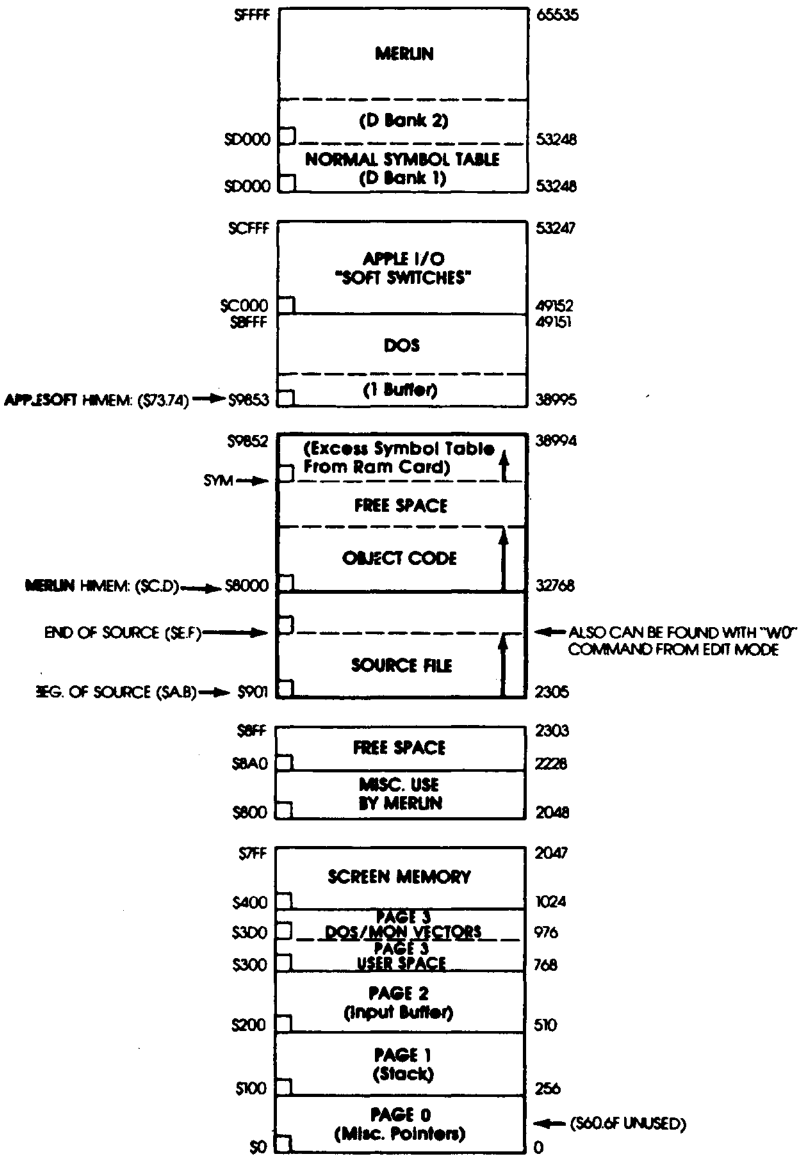
若進入了 BASIC 可用“ASSEM”重新進入 MERLIN，或者用BRUN MERLIN 或由磁碟重新載入也會執行溫起動(WARM START)，如果MERLIN 還在的話，不再載入MERLIN。

或者可用BRUN MERLIN ASM強迫重新做冷起動(COLD START)，同時清除掉所有在記憶體中的程式。

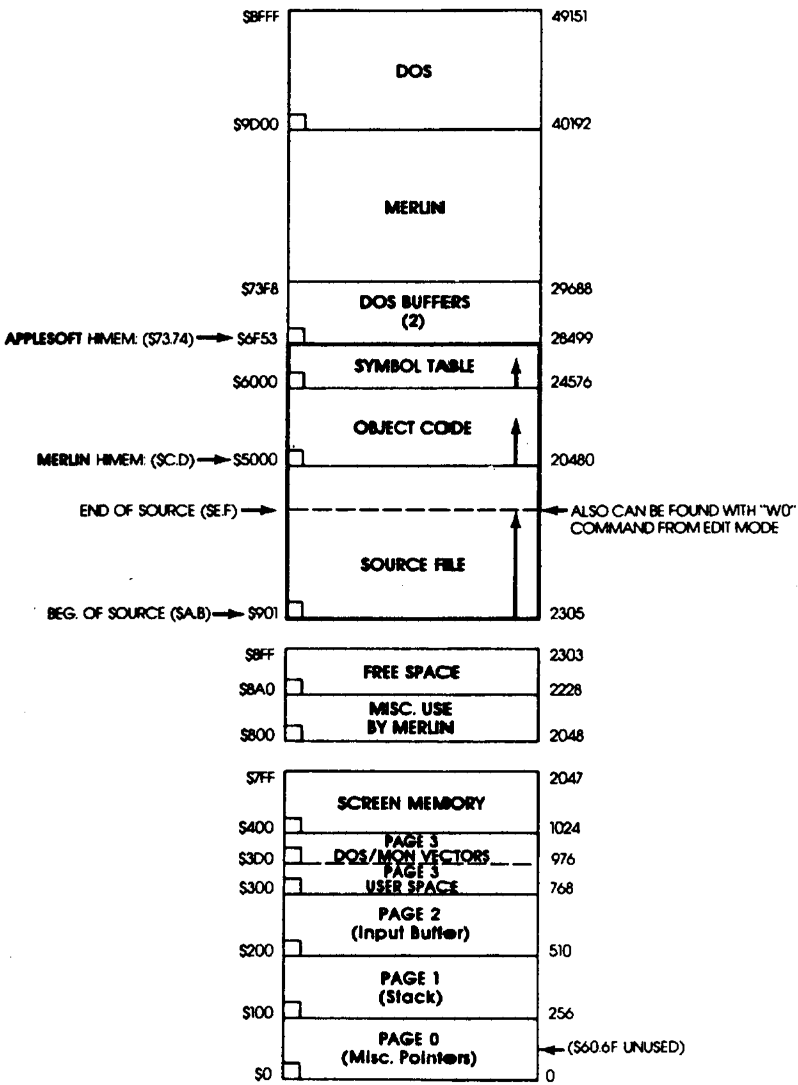
一般大小的程式，使用者不須去煩惱記憶體如何使用，但大程式則需要知道某些限制。HIMEM (假設爲 $8ØØØ) 爲原始程式的上限，也是 PUT 程式的上限。若執行組合程式時在 PUT 的列號發生錯誤，表示 PUT 的程式超過HIMEM，則必須增加HIMEM。 ORG 及 OBJ 假設的位址也是在HIMEM。OBJ 位址不得低於 HIMEM ，除非是在第三頁。若 OBJ 在第三頁，使用者必須注意程式不可大到破壞DOS 在 $3DØ－$3FF 的 JMP 指令。因爲MERLIN並沒有檢查這個錯誤。

若在組合程式執行時，目的碼超過BASIC HIMEM 上 (或 SYM位址以上) 則目的碼不再寫入記憶體，但組合仍然繼續進行，且其輸出送到螢幕或印字機，如此唯一的作用是使得 SAVE 指令失去作用，所以當你需要一份很長的程式列印時，可以超過DOS 所在的位址 (但不產生目的碼)。

8.2 MERLIN記憶體配置圖（RAM CARD版）



8.3 MERLIN 記憶體配置圖(48 K 版)



8.4 符號表

除非LST OFF否則組合結束後會列出符號表，首先按字母排序，然後按位址數字排列，並且每個符號皆有標示：

MD = 集體指令定義
M = 集體指令中用到的標題（如 LOOP，OUT）
V = 變數（包括] 的符號）
? = 定義而未用到的符號

內部的做法是以符號長度位元組的第 7 到第 4 位元代表每一種狀況：（ 1 代表眞）

? = 位元 7　MD = 位元 5　M = 位元 4

而位元 6 在以字母排列印出時設爲 1 ，以位址數字印出時清爲零。符號表的印出格式是以 80 字印字機的格印，或是 40 字印字機而其能在超過 40 字時自動執行送行（ LINE FEED）。

8.5 移位鍵在MERLIN下的用法

MERLIN 提供硬體移位鍵的功能，CONFIGURATION 程式（附在系統磁片上）可提供此功能，並且能自動設定移位的功能，若沒有硬體的移位修正則自動設定移位無效。所以可以在任何一種機器上執行。

8.6 80行卡的用法

大部份的人都擁有8Ø行卡，可用VIDEO指令起動8Ø行卡。若想在一開機就自動起動8Ø行卡可用CONFIGURATION程式。在RESET後VIDEO Ø或VIDEO$1Ø 的指令可回到正常APPLE的螢幕。

若你的8Ø行卡沒有反轉（INVERSE）輸出功能，則控制字母看起來就是正常字而不是反轉字。可用FIND指令尋找是否有控制字母，或是回APPLE 正常螢幕。

如果你的MERLIN已設定8Ø行卡在第三SLOT，而SLOT上並沒有8Ø行卡，則MERLIN自動去除8Ø 行卡的功能。所以換機器時無須再重執行CONFIGURATION。若有些80行卡，不能使用"POKE 36"來做表格定位的話，則不能在MERLIN上使用。據我們所知，目前這類的80行卡，只有舊版的FULL VIEW 80行卡。

在EDIT模式下工作時，MERLIN控制所有的I/O開關，所以控制碼的功能不是如8Ø行卡所說，而是執行MERLIN的功能。例如CTRL-L並不執行清除螢幕，而是轉換大小寫。再如CTRL-A 並不執行大小寫轉換而是在檔案列裏放進一個CTRL-A的字母（$80）。

8.7 Configure ASM 程式說明

這個程式讓你能對MERLIN的一些假設值做少許的修改。

1. 當進入ASM時會問"UPDATE SOURCE"所搜尋的字母（原爲"/"）可更改。
2. EDIT模式下的外卡（wild card）字母（原爲^）。

3. 符號表列印出時每行的個數(原來每行四個)。
4. 是否有 8Ø 行卡，在那一個 SLOT。
5. 是否有小寫轉換器。
6. 設定所使用印字機的品牌。

最後可把修改的程式存入磁片(或是另外一片);不需要保留原來的版本，因為只要再重新執行一次 CONFIGURATION 就可改回原來的版本。

程式最後可選擇要不要把"MERLIN"這程式存到別的磁片，因為普通的方法不能做到，可用 FID把 ASM. OBJ 及 MERLIN轉到其他磁片。

注意：MERLIN只能在原來保護的系統磁片上執行，轉到其他磁片只是為了安全而已，除非重新拷貝回原來的磁片，否則無法執行正常功能。CONFIGURATION只能在 APPLE 原來的螢幕下 BRUN。

8.8 錯誤信息

8.8.1 錯誤的運算碼

當運算碼不正確(可能是拼錯字)或者當運算碼位於標題欄的位置。

8.8.2 錯誤的定址模式

定址模式不是 6502 的指令，例如 JSR(LABLE)，或 LDX(LABLE)，Y。

8.8.3 錯誤的分歧

分歧超過± 127 位元組。

8.8.4 錯誤的運算元

不合規定的運算元格式。在已出現過的標題之後以"EQU"定義標題爲零頁，也會發出此錯誤信息。運算元超過 6 4 位元組，或註譯行超過64 位元組也會發生此錯誤。這種錯誤會使組合程式中斷。

8.8.5 符號重覆定義

發現兩個相同的符號。

8.8.6 記憶體已滿

這通常是由以下四種原因所引起：OBJ 設定錯誤，原始程式太長，目的碼太長，或符號表太大。參閱本章最後"特別注意"。

8.8.7 標題未定義

你的程式用到某個標題而此標題未定義，或者用 PMC 以外的指令呼叫集體指令。標題在"條件組合"爲OFF 之下定義。但後者對集體指令的標題是可以的。

8.8.8 不是集體指令

向前呼叫集體指令，或者 PMC 呼叫的標題不是集體指令。

8.8.9 巢次錯誤

集體指令呼叫集體指令層次超過 15，或條件組合層次多過 8 層。

8.8.10 PUT不正確

原因是集體指令定義 PUT，及 PUT的檔裏用到 PUT指令。

8.8.11 SAV錯誤

原因是集體指令裏用到“SAV”，或是從上次SAV 後用到了一個以上的 OBJ 指令。

8.8.12 輸入錯誤

若KBD指令用以輸入一個標題的值時，沒有輸入或輸入超過 37 個字母，將會發生錯誤。

8.8.13 中止

當ERR指令的運算元不為零時，發出此錯誤。

8.8.14 標題錯誤

使用EQU或MAC指令時沒有標題、標題太長、標題有不合規定的字母。

8.9 特別注意事項—記憶體已滿錯誤

通常產生此錯誤的原因有四，詳細的說明如下：

1. “MEMORY FULL IN LINE：n”記憶體已滿於第n行。發生在第一階段時，列號指於一個OBJ的指令。原因：OBJ的位址低於HIMEM（通常爲$8ØØØ）且不在第三頁。MERLIN不允許將程式放在這些位址以免破壞你的原始程式及作業系統。解決方法：把OBJ指令去掉或者把OBJ的位址放於正確的位址。
2. “ERROR：MEMORY FULL”記憶體已滿。發生在輸入一行很長的資料行時。原因：你的原始程式已超過HIMEM。解決方法：把HIMEM設定到更高的位址，或者把原始程式分段，以後再用PUT一個一個叫進來。
3. “MEMORY FULL IN LINE：n”記憶體已滿於第n行。發生在第一階段結束開始組合時。原因：符號及標題太多，以致超過APPLESOFT的HIMEM（RAM CARD版爲$9853）。解決辦法：用SYM指令把符號表的起始位址降低，以增加記憶體空間。
4. 沒有任何的錯誤信息，但並沒有產生目的碼（在EXEC模式下沒有目的碼的顯示）。原因：目的碼蓋到符號表或者超過APPLESOFT的HIMEM。解決辦法：把MERLIN的HIMEM降低，以儲存

更多的目的碼，或者用DSK指令直接把目的碼寫到磁碟去。

9. SOURCEROR

9.1 簡 介

SOURCEROR 是一個複雜的而容易使用的反組合器（disassembler）。對於二進位的程式（目的碼），作爲轉換成 MERLIN原始檔案的輔助工具。 SWEET 16 碼和 65Ø2 碼都可適用。

SOURCEROR的主要部分是 SRCRR. OBJ, 但是這個程式不能直接去執行（一般狀況下）。因爲它會蓋掉DOS的緩衝區，最後破壞系統。基於這個原因，提供了一個小程式叫SOURCEROR ，它在輸入緩衝區執行，而不會和其他記憶體裏的程式相衝突。這個小程式只是做記憶體大小的檢查；去除可能和SOURCEROR主程式衝突的程式（設定 MAXFILES 1），如PLE；然後執行SRCRR. OBJ（在$ 88ØØ－$ 9AA5）。

爲減低意外發生的可能性到最小，SRCRR. OBJ有一個假設位址 $ 4ØØØ，當你執行（ BRUN ）它後，它會不做任何事而返回。如果你嘗試在實際的位址 $88ØØ 去執行，你可能會陷入大痲煩之中。SOURCEROR假定使用標準

Apple 螢幕，而不會在 8Ø 行卡上工作。

9.2 使用SOURCEROR

1. 將需要反組合的程式載入。雖然SOURCEROR 可以處理任何位址的程式，但是該程式的原始位址最好不要和SOURCEROR 和其所產生原始檔案的位址相衝突。當你有疑問時，載入該程式到 $800 或 $803 。對在 $4000 以上的小程式，或在 $6000 以上的中等程式，以它們原來的位址就可以了。
2. BRUN SOURCEROR
3. 你將會被告知原始檔案的假設位址在 $2500。之所以選擇這個位址的原因是：不會衝突到大多數你想要反組合的二進位程式。鍵入RETURN，接受假設位址；或者，指定你所要的位址（十六進位）。在這個時候，你也可以得到一個“秘密的”規定，亦即鍵入CTRL-S在原始位址之後，或者取代原始位址。於是系統會要求你指定一個（非標準的）位址給SWEET 16 解釋器使用。這是用來反組合以RAM版本 SWEET 16 編寫的程式。
4. 接著鍵入RETURN，表示程式以其原始（執行）位址來反組合；否則，你必須指定被反組合碼的現在位址（十六進位）。最後，你會被要求給予該程式的原始位址。

   在程式反組合時，你必須使用程式的原始位置，而不是程式目前所在的記憶位址。這就好像是在

原始位址將程式反組合一樣，而實際上，SOURCEROR是把現在位址的碼反組合，然後轉換到原始位址去。

5. 最後，螢幕會顯示一頁表頭，包括在反組合所用命令的摘要。現在你可以開始反組合程式或是使用其他的任何命令。你第一個命令必須包括一個十六進位的位址，以後則可選擇。

在這個時候，或者到了處理結束時，你可以鍵入RESET回到SOURCEROR的起始處。如果再鍵入一次RESET，你會離開SOURCEROR而回到BASIC。使用RESET鍵時，系統是假設你的Apple是自動起動（ autostart ）監督程式的版本。

9.3 反組合命令

反組合命令和Apple監督程式的命令相似。所有的命令都接受命令字母前的四個數字的十六進位位址。假如這個數值被省略，就會從現在的位址起反組合。開始進入時，必須要指定一個數值。

如果你指定的數值超過現在的位址，就會產生一個新的ORG。

一般而言，如果你指定的數值小於現在的位址；在這種情況下，反組合器會檢查是否等於前一次的位址。是的話，就直接回到那位址，否則就回到下個被用到的位址，並且產生新的ORG。後來的原始列被“消除”。通常儘可能最

好避免產生新的ORG。你如果得到一個新的ORG，而又不想要時，可以一點一點後退，直到在反組合時不再得到新的ORG為止。

9.4 命令描述

9.4.1 L (List)

這是反組合的主要命令，它會反組合20列目的碼。它也可以重複使用（例如：2ØØØLLL會從$2ØØØ開始反組合60列）。如果發現到JSR到SWEET 16 解釋器時，會自動切換到SWEET 16的模式。

L命令經常保持在目前反組合的模式（SWEET 16 或正常）。如果碰到非正常的碼時，會響鈴一聲並且列出三個閃爍的問號。這只是在提醒你這種情況而已。對目的碼本身而言，不承認的運算碼被轉換HEX資料，但不顯示在螢幕。

9.4.2 S (SWEET)

和L相似，但是強迫反組合程式為SWEET 16 模式。當碰到SWEET16的RTN運算碼，就返回正常的56Ø2模式。

9.4.3 N (Normal)

和L相同，但強迫反組合程式為正常65Ø2模式。

9.4.4 H (Hex)

產生HEX資料的運算碼。假設值為一個位元組。如果

你在H後面加上一個位元組（一個或兩個數字）的十六進位數值，那麼就會產生相同數目的位元組資料。

9.4.5 T (Text)

這是將現在位址的資料反組合爲ASCII 字串。視資料型式的不同，會反組合下列的虛運算碼：ASC, DCI, INV 或FLS。適當分界號也會自動的加上去。當遇到不正確的碼時，或者已經處理了62個字元，或者資料的高位元改變時，反組合都會自動結束。在最後一種情況下，ASC運算碼會自動變爲DCI。

有時轉變爲DCI也是不適當的。可以藉著TT 代替T命令，來廢止這種轉變。

有時候，由於接下來的碼，對SOURCEROR言，類似ASCII碼，那麼反組合的字串，就不會在適當的位置停住。在這種情形下，你可以在T命令的後面加上一個或二個數字的十六進位值來限制反組合字串的長度。

T或TT命令，也可以用來在正規的ASCII 字串，和閃爍的ASCII 字串間建立正確的界限。這兩者的界限通常是很明顯的。任何小寫字母出現在字串中，都會以閃爍的大寫字母顯示。

9.4.6 W (Word)

將現在位址的下面兩個位元組的資料，反組合成DA運算碼。另一種命令型式如WW，就會反組合爲DDB 的運算碼。最後，如果使用W－做爲命令的話，兩個位元組就會被反組合成DA LABEL-1 的型式。後者常是把位址放進堆疊（STACK）裏的正確型式。你可以在反組合時就偵測到

，或者在程式已經反組合後才發現。後者有助於你再反組合時，記下該注意的地方。

9.5 內部命令

9.5.1 / (Cancel)

基本上，取消上一次的命令。更正確地，它重新建立上一次設定的位址（命令所用位址並不需要眞正附帶一個位址）。當需要後退時，可以很方便地讓你不用去管剛才鍵入的位址。例如：假設你鍵入T去反組合某些文字資料，但你又不知道下面的資料，於是你只要鍵入L，去看看下面的資料。如果接下的本文資料是以一些十六進位資料（例如$8D carriage return ）作結束時，你只要鍵入“／”，來取消L命令，然後鍵入適當的H命令。

9.5.2 R (Read)

這讓你以另一種型式來看記憶體的內容，使得隱藏的本文資料顯示出來。例如要看 $1234 到 $1333 之間的資料，鍵入1234 R；接著再單獨鍵入R，下一頁記憶的資料就會顯現出來。這個命令所用的數值和反組合的位址是完全無關的。

再者，你可以這樣反組合，先鍵入（位址）R，然後再鍵入單獨的L，那麼反組合會依照原來的位址列印，就好像你從未用到命令R一樣。你如果不想在回到反組合程式時，使用假設的位址，你可以在要返回地方做個記號，或者先使

用／，再用R命令。

9.5.3 Q (Quit)

這個命令結束反組合程式，並且自動進入最後處理程序。如果你在Q之前鍵入一個位址，那麼在最後處理程序之前，位址指標會退到該位址（但不包括該位址）。例如：在反組合列印的最後出現下列數列：

```
2341-   4C   Ø3   EØ          JMP   $EØØ3
2344-   A9   BE   94          LDA   $94BE,Y
```

最後一列是沒有用的，所以鍵入 2344Q。如此，會取消最後一列，而保留了前面一列。

9.6 最後處理程序

在命令Q以後，會把反組合碼做最後幾分鐘的處理。如果在此時，你鍵入 RESET，你會回到 BASIC 而且失去反組合碼。

對於短的程式，最後處理可能只需要一兩秒鐘；但對於長的程式，可能會耗費兩三分鐘。耐心等著吧！

當處理完畢後，會詢問是否要把原始碼儲存起來。如果是的話，會要求一個檔案名稱。SOURCEROR 會在檔案名稱後加上“.S”，然後存到磁碟裏。

儲存檔案所用的磁碟機與BRUN SOURCEROR的相同。如果你要把原始檔案存到另一片磁碟的話，先改變磁碟片。

若要看反組合後的檔案，只需BRUN MERLIN 或者鍵入ASSEM，然後將檔案載入即可。

9.7 完成後檔案的處理

在大多數的情況下，當你有些經驗及很小心的使用後，原始檔案應該只有一點或沒有缺點。

你可能發現某些DA的運算碼，應該改爲DA LABEL −1或者是DDB LABEL 的型式。在這種或類似的情況下，你最好把該注意的地方記下，然後重新反組合一次。如果你要適當地改變原始檔案的話，重新反組合一次會顯得更容易，更快也更無痛苦。

在原始檔案EQU表中，會包括所有外部的或其他不被承認的標題。你要很仔細地查看這個表格。除了是由DA，JMP或JSR所得的結果，否則此表中不能包括任何零頁的相等值。這幾乎是個很肯定的記號，表示在反組合時的錯誤（你自己的，而非SOURCEROR的）。它可能發生在把一個資料區當做正常的碼來反組合。

注意：在這種情況下，如果你想嘗試組合原始檔案，只要這些相等值出現，你就會得到錯誤的訊息。如果你把EQU表移到程式的開始（正常下你應該如此），你可能不會得到錯誤訊息，但是組合的結果可能不會正確。當麻煩出現時，最重要的就是先處理這種狀況。例如：反組合器找到如下的資料：ADØØ8D ；它會正確的反組合爲：LDA $ØØ8D。

而組合器總是會將這個碼組合爲零頁指令，而成爲兩個位元組：A5 8D。你經常會發現一

個程式中，使用這種型式的零頁次指令。在那種情況下，你必須在LDA運算碼後插入一個字元，使得組合程式和原來的一樣。通常，資料出現在運算碼的前面，於是更必須加以處理，以便得到正確的組合。

9.8 記憶已滿訊息

當原始檔案到達距離SOURCEROR起始處$6ØØ（也就是檔案超過$8200）時，你會看到閃爍的“MEMORY FULL”及“HIT A KEY”的訊息。當你按下一個鍵後，SOURCEROR會直接進入最後處理程序。保留$600位元組的原因是：SOURCEROR需要一塊地方來做最後處理之用。在最後處理程序時，可能（並非多半）SOURCEROR的一部份會被蓋掉，但這並不會產生問題。因為在那時候，SOURCEROR不會再到那地方了。這裏有一個“秘密的”儲備，使得MEMORY FULL被忽視。如果此時你鍵入CTRL-O的話（表示忽視），於是SOURCEROR會回覆另一個命令，你可以利用這個命令來指定所需要的結束位址。你可以利用它來使得SOURCEROR多反組合幾條指令。但很明顯地，你不能用得太多。

· 注意：在離開SOURCEROR後，千萬不要利用CALL來重新執行SOURCEROR程式，你要重新從磁碟上載入再執行。這是因為DOS緩衝區在離開時，已經被重建，而且可能損壞部分的SOURCEROR。

9.9 LABELER程式

SOURCEROR 的優點之一是可以在反組合時，對於那些承認的位址自動的予以設定標題。所謂被承認的位址是指在 SOURCEROR的標題表中所定義的標題位址。例如：所有的 JSR $FC58 會反組合爲 JSR HOME。這個標題位址表是可以利用 LABELER 來加以修改。

BRUN LABELER 執行後，程式會注意 SRCRR.OBJ 是否已經載入到記憶體，然後進入主程式功能表。

9.10 LABELER的命令

9.10.1 Q: QUIT

當你結束標題表的修改後，鍵入'Q'而離開 LABELER 程式。如果你想要把你的檔案儲存起來，則鍵入'S'。否則鍵入ESCAPE，離開而不儲存新的表格，例如：你只是在查看這些表格。

9.10.2 L: LIST

這使得目前的標題表條列出來。在鍵入'L'後的任何按鍵會開始列印。以後任何鍵都會顯示下一頁的標題表；CTRL-C則會結束列印。

9.10.3 D : DELETE LABEL(S)

使用這個選擇來刪除你所不需要的標題。在鍵入D命令後，只要很簡單地輸入該標題的編號即可。如果你要刪除一個區域，那麼輸入一個起始和終止的標題編號，中間以逗號隔開即可。

9.10.4 A : ADD LABEL

利用這個選擇來增加一個標題到表格中。只要告訴程式的十六進位位址以及你想要使用的名字，然後鍵入RETURN來結束這個命令。

9.10.5 F : FREE SPACE

這會告訴你表格中還剩下多少的自由空間可供你輸入新的標題。

9.10.6 U : UNLOCK SRCRR.OBJ

在儲存新的表格檔案前，你可能需要把SRCRR.OBJ加以解鎖。如果你要儲存新的表格檔案時，在結束（Q）LABELER 程式之前，使用這個命令。

10. SWEET 16簡介

作者：Dick Sedgewick

SWEET 16可說是Apple II最少被用到並且最少人瞭解的程式。

如同整數培基（INTEGER BASIC）及APPLESOFT培基一樣SWEET 16是一種電腦語言，它較培基語言更接近65Ø2的組合語言。

要使用SWEET 16語言，必須先學習這種語言。它的運算碼很少且不難學習；SWEET 16是由Steve Wozniak先生——蘋果公司創始者所發展的。

SWEET放在所有APPLE II（不是現在通行的Apple II+及APPLE IIe）的僅讀記憶體上，位址由$F689到$F7FC。它有獨立的一套運算碼及指令群，利用監督程式裏的SAVE及RESTORE副程式以保存65Ø2的暫存器，所以SWEET 16可像副程式來使用。

它用零頁最前面的32個位元組（$0～$1F）當作它的16個雙位元組暫存器，所以除非做些許的改變，無法與APPLESOFT培基共通（compatiable）。

最早的一份文書報告有關SWEET 16是在BYTE雜誌

1977 年 11 月份的一篇文章“ SWEET 16: THE 65Ø2 DREAM MACHINE”，後來出現在“ WOZ PAK”上。這篇文章拿來當作例子以幫助瞭解 SWEET 16 ，並說明其用法。

使用 SWEET 16 的程式如 PROGRAMMER'S AID #1 ，中的 Renumber（重排序號），APPEND（附加），RELOCATE（重新定址）等的程式中。而 PROGRAMMER AID 的使用手册中有詳細的 SWEET 16 原始程式，參見該書第 65 頁。

示範程式包含三個部份，非常簡單：

1. 整數培基程式
2. 機器語言程式
3. SWEET 16 程式

此程式的目的是搬動記憶體的資料。搬動的參數由整數培基輸入。

列號 12Ø 的“ CALL 768 ”（ $3ØØ ）的作用是進入 SWEET 16 並返回培基（ $3ØØ, $3Ø1, $3Ø2 及 $312 ）。SWEET 16 副程式所做的工作當然是搬動資料，其進入點在 $3Ø3 到 $311 （請看列表 # 3 ）。

搬動後，螢幕將會顯示三列資料，每列 8 個位元組，並等待輸入另一組參數。這三列為：

第一列：由 $8ØØ 開始的 8 位元組資料，為所要搬動的資料起始位址。

第二列：由所輸入的目的位址開始的 8 位元組資料。

第三列：由 $ØØØØ 起的 8 位元組資料。（ SWEET 16

最前面的四個暫存器)。

顯示這三列的資料以說明工作進行的情形。

整數培基有自己的一套方式儲存字串"A$",用2個位元組儲存字串的名稱,在字串的資料之前共有5個位元組表示字串的一些規格,並在字串最後有一個字串"結束記號"。

爲了顯示方便並能在第8位元組看到"字串結束記號",所以由鍵盤輸入的資料A$(最好不要超過二個字母),會顯示在第六及第七位元組。

然後目的位址也要鍵入,爲了方便只要輸入高位元組即可,本示範程式限制此高位元組不得小於9,不得大於144(48K APPLE)以避免因搬動而破壞程式。

第三列的顯示資料,顯示SWEET 16四個暫存器,其目的是顯示每次搬動後此四個暫存器的內容以讓使用者了解SWEET 16的工作情形。

由"SWEET 16: The 65Ø2 Dream Machine"文章裏可知道它用到最開始的16個雙位元組,也就是$ØØ 到$1F的零頁記憶體。

程式中搬動的過程可由前四個暫存器詳細的看出,所以顯示8個位元組資料就足以說明其工作。

此四個暫存器爲RØ, R1, R2, R3:

RØ	$ØØØØ	及	ØØØ1	SWEET 16的主暫存器(Accumulator)
R1	$ØØØ2	及	ØØØ3	來源位址
R2	$ØØØ3	及	ØØØ4	目的位址
R3	$ØØØ4	及	ØØØ5	搬動位元組的數目
.				
.				
.				
R14	$ØØ1C	及	ØØ1D	儲存上次所得結果的暫存器
R15	$ØØ1E	及	ØØ1F	SWEET 16程式計數器

另外R14，R15的用法在"WOZ"文章裏也舉例說明其用法，以幫助了解SWEET 16。注意R14的高位元組（位於$1D)內容爲$Ø6，存的是暫存器的位址（3 * 2 = 6），R15是SWEET 16的程式計數器內容指向下一個要執行指令（SWEET 16的指令），當示範程式結束時其爲$312，之後緊跟著的就是65Ø2的機器碼。

開始示範程式之前先把LISTING #1的培基程式輸入電腦，當然REM指令可省略。列號10只在有磁碟機時才有用，可先由磁碟機載入機器碼。LISTING #2也要輸入於$3ØØ的位址。

注意：65Ø2的反組合程式並不能把LISTING #3顯示出來，但可用SOURCEROR反組合器來產生LISTING #3。

先輸入"RUN"，再按（RETURN）
鍵入"12"，再按（RETURN）（A$的資料）
鍵入"18"，再按（RETURN）（目的位址的高位元組）

完成之後，螢幕將顯示：

```
$Ø8ØØ-C1 4Ø ØØ 1Ø Ø8 B1 B2 1E （來源）
$ØAØØ-C1 4Ø ØØ 1Ø Ø8 B1 B2 1E （目的）
$ØØØØ-1E ØØ Ø8 Ø8 Ø8 ØA ØØ ØØ (SWEET 16)
```

注意：$AØØ裏的8位元組與$8ØØ裏的8位元組相等，表示已正確的搬動了8位元組。它們一次一個位元組由代用值（TOKEN）$C1起到$1E結束地搬。如果搬動資料個數少於8則其後的資料

將不會改變。

8個位元組各有下列意義：

```
C1      40   00   10    08     B1       B2          1E
--------- ---- ------- ------- ---------           --
    |         |      |          |         |
變數名稱     DSP    NVA        資料      資料     字串結束碼
```

SWEET 16暫存器顯示如下：

```
        低       高      低       高      低       高      低       高
$0000   1E       00      08       08      08       0A      00       00
       -----------     ----------      ----------       ----------
           |               |                |                |
           R0              R1               R2               R3
        主暫存器         來源位址         目的位址      （位元組個數）
```

R0的低位元組爲$1E，表示最後一個搬的資料爲$1E。

R1的低位元組開始爲零，每搬一個位元組就增加1，所以最後是8。

R2的高位元組爲$0A，也就是由鍵盤輸入“10”，並放入SWEET 16的碼中。低位元組與R1的低位元組一樣的情形。

R3裏原先放8，爲搬動的個數，每搬一個就減1，所以減至零。

每次輸入不同位資料就可看出SWEET 16暫存器是如何變化的。

如想要用SWEET 16寫程式，可詳讀下一章所附的文章，並多執行此示範程式可幫助你處理16位元的資料。

SWEET 16爲一種語言（或工具），爲APPLE Ⅱ的

使用者提供一個新的方向，而且一直都在你的APPLE Ⅱ上，不須多花一毛錢。市面上流行的爲APPLE Ⅱ+，並不包含INTEGER BASIC，但只要用DOS3.3系統磁片開機後，會自動把INTEGER BASIC載入RAM CARD，執行INT指令則自動轉爲INTEGER BASIC。或者MERLIN磁片上附有SWEET 16的原始程式。

APPLE的使用者若想學習機器語言，可先從SWEET 16著手，因爲SWEET 16的指令群較易學習，可以比較快進入狀況。

10.1 列表＃1

```
>List
    10      PRINT "[D]BLOAD SWEET":  REM CTRL D
    20      CALL - 936:  DIM A $ (10)
    30      INPUT  "ENTER STRING A $ " , A $
    40      INPUT  "ENTER # BYTES " , B
    50      IF NOT B THEN 40 : REM AT LEAST 1
    60      POKE  778  B : REM POKE LENGTH
    70      INPUT  "ENTER DESTINATION" , A
    80      IF A > PEEK (203) - 1 THEN 70.
    90      IF A < PEEK (205) + 1 THEN 70
    100      POKE  776 , A : REM POKE DESTINATION
    110      M = 8 :  GOSUB 160 :  REM DISPLAY
    120      CALL  768 : REM GOTO $0300
    130      M = A :  GOSUB 160 :  REM DISPLAY
    140      M = O :  GOSUB 160 :  REM DISPLAY
    150      PRINT :  PRINT :  GOTO 30
    160      POKE 60 , 0 :  POKE  61 , M
    170      CALL  -605 :  RETURN  :  REM  XAM8  IN MONITOR
```

10.2 列表＃2

```
300:20  89 F6 11  00  08  12  00  00  13  00  00  41  52
F3  07 FB 00  60
```

10.3 列表＃3

```
SWEET 16

     $300  20  89  F6  JSR    $F689
     $303  11  00  08  SET    R1  source address
     $306  12  00  00  SET    R2  destination address
                        A
     $309  13  00  00  SET    R3  length
                   B
          $30C  41            LD     @R1
          $30D  52            ST     @R2
          $30E  F3            DCR    R3
          $30F  07            BNZ    $30C
          $311  00            RTN
          $312  60            RTS
```

Data will be poked from the Integer Basic program:

"A" from Line 100
"B" from Line 60

11. SWEET 16—虛擬的16位元微處理機

11.1 簡 介

當我在為 65Ø2 微處理機寫 APPLE 的培基時， 一再碰到各種的“墨菲定理”(MURPHY'S LAW，譯者註：墨菲定理簡略的說是軟體之中如果有錯誤，一定不只一個。一塊麵包掉到地上，一定是塗奶油的那一面朝下，碰到你的長毛地毯。)簡單的說，任何處理 16 位元資料的工作總需要二倍的程式。任何用到大量 16 位元指標的程式(如編譯程式，編輯程式，組合程式)都屬於這種情形。以我的情形來說，即使 65Ø2 能增加些許雙位元組運算的指令也只能稍微解決問題。我眞正需要的是一個 6502/RCA 18ØØ 二種微處理機的混合體，有很多的 16 位元暫存器，和極佳的指標功能。我的解決辦法是創造一個虛擬的 16 位元微處理機，使用解釋程序(Interpreter)，我叫它做 SWEET 16。

SWEET 16 的基礎是建在 16 個 16 位元暫存器上(RØ—R15)，實際上是 32 個記憶體的位址。RØ做為 SWEET 16 的主暫存器(ACC)， R15 為程式計數器(PC)， R14 做為旗標暫存器，R13保留比較指令的結果，R

12 是副程式返回堆疊的指標。其餘的暫存器可給使用者做各種用途。

SWEET 16 的指令分爲兩種：一爲暫存器的指令，另一爲非暫存器的指令。暫存器的 指令其運算碼指定 16 個暫存器其中的一個做爲資料，或是資料的指標，完全視何指令而定，例如 INR R5，就是把R5 當作資料，ST @R7就是把R7 當作資料的指標。除了 SET指令所有的暫存器指令皆爲單位元組。非暫存器指令皆像6502 的分歧指令，第二碼存放 + / - 127 位元組的位移。如果上一個暫存器指令結果符合分歧的條件，則把第二碼加到SWEET 16的 PC，產生分歧的效果。

SWEET 16 可做爲一種 6502 擴充軟體套裝程式，而並非一個獨立的處理器。當 6502 執行一個呼叫 SWEET 16 的指令後就進入 SWEET 16 的工作模式，而接下去的指令碼就被當作 SWEET 16 的運算碼。而當 SWEET 16 執行到非暫存器指令 RTN 則工作模式又再轉回 6502 。同時會把 6502 的暫存器（ A, X, Y, P, S ）取回。底下的例子說明如何使用 SWEET 16：

```
300  B9 00 02            LDA   IN,Y      取一個字母
303  C9 CD               CMP   #"M"      “M”則做搬動資料
305  D0 09               BNE   NOMOVE    不是，跳過
307  20 89 F6            JSR   SW16      是，呼叫 SWEET 16
30A  41         MLOOP    LD    @R1       R1 存放來源資料位址
30B  52                  ST    @R2       R2 存放目地資料位址
30C  F3                  DCR   R3        長度減 1
30D  07 FB               BNZ   MLOOP     分歧回去，一直到完成
30F  00                  RTN             回到6502工作模式
310  C9 C5      NOMOVE   CMP   #"E"      是否爲 E
312  D0 13               BEQ   EXIT      是的，結束
314  C8                  INY             不是，繼續
```

注意：暫存器A, X, Y, P, S 不會被SWEET 16改變。

11.2 指令介紹

SWEET 16指令群非常短而且不複雜。除非分歧指令比較麻煩一些，其它指令甚至可以用手組合。所有的暫存器指令由兩個16進位的數字組成（ø～F），一個是運算碼，一個指定16個暫存器中的1個。例如15和45都是R 5的指令，而23, 27, 29都是ST 指令。所有暫存器指令都設定爲成雙的，以方便記憶，所以L D和S T各爲2 N和3 N，LD@和ST@各爲4 N和5 N。

ø到$C的運算碼指定爲13個非暫存器指令。除了RTN（opcode ø），BK(øA)，RS(øB)，其餘都是6502型式的分歧。第二碼爲分歧的位移值（＋／－127），以2的補數格式，相對於下一個指令。

如果指定的分歧條件符合，則第二碼就被加到P C裏，導致分歧的效果。除了B R（無條件分歧），B S（分歧到副程式），所有的分歧指令都設定爲成雙的，以方便記憶及用手組合。例如B P（正數則分歧），B M（負數則分歧）的運算碼爲4和5，B Z（零則分歧），BNZ（非零則分歧）的運算碼則爲6和7。

BaBel II

11.3 SWEET 16運算碼總說

11.3.1 暫存器指令

1n	SET	Rn	常數（設定）
2n	LD	Rn	（載入）
3n	ST	Rn	（儲存）
4n	LD	@Rn	（間接載入）
5n	ST	@Rn	（間接儲存）
6n	LDD	@Rn	（間接雙位元組載入）
7n	STD	@Rn	（間接雙位元組儲存）
8n	POP	@Rn	（間接堆入）
9n	STP	@Rn	（間接取存）
An	ADD	Rn	（加）
Bn	SUB	Rn	（減）
Cn	POPD	@Rn	（間接雙位元堆入）
Dn	CPR	Rn	（比較）
En	INR	Rn	（加1）
Fn	DCR	Rn	（減1）

11.3.2 非暫存器指令（ea：爲有效位址，+/－127）

ØØ	RTN		（回到6502工作模式）
Ø1	BR	ea	（無條件分歧）
Ø2	BNC	ea	（無溢位則分歧）
Ø3	BC	ea	（溢位則分歧）
Ø4	BP	ea	（正數則分歧）
Ø5	BM	ea	（負數則分歧）
Ø6	BZ	ea	（零則分歧）
Ø7	BNZ	ea	（非零則分歧）
Ø8	BM1	ea	（負1則分歧）
Ø9	BNM1	ea	（非負1則分歧）
ØA	BK		（中斷）
ØB	RS		（副程式返回指令）
ØC	BS	ea	（分歧到副程式）
ØD			（未定義）
ØE			（未定義）
ØF			（未定義）

11.4 暫存器指令

11.4.1 設定

SET Rn，常數 1n 低 高

雙位元組的常數放到Rn裏（n = Ø到F，16進位），並設定所有分歧條件，溢位旗標清為零。

例子：

```
15 34 AØ   SET R5, $AØ34     (R5內容被填為$AØ34)
```

11.4.2 載入

LD Rn 2n

主暫存器由Rn載入資料，並設定所有分歧條件，溢位旗標清為零，Rn的內容不改變。

例子：

```
15 34 AØSET R5, $AØ34
25      LD R5                ACC 內容為$AØ34
```

11.4.3 儲存

ST Rn 3n

ACC的內容儲存到Rn，分歧條件皆被設定，溢位旗標清為零，ACC的內容不變。

例子：

```
25   LD R5
36   ST R6                   把R5的內容放進R6。
```

11.4.4 間接載入

```
LD @Rn  4n
```

ACC 低位元組的資料被塡爲Rn 指標所指的記憶體位址內容。ACC高位元組清爲零，分歧條件依 ACC 最後的內容設定，所以一定是正數，且一定爲非負1，此指令執行後Rn自動加1。

例子：

```
15 34 AØ SET R5, $AØ34
45          LD  @R5
```

$AØ 34記憶體內容放進ACC低位元組，ACC高位元組爲Ø，R5加1成爲$AØ 35。

11.4.5 間接儲存

```
ST @Rn   5n
```

ACC 低位元組的資料存入 Rn 指標所指的記憶體裏。分歧條件依ACC 的第 2 個位元組資料設定，溢位旗標淸爲零。。此指令執行後 Rn 自動加1。

例子：

```
15 34 AØ   SET    R5, $AØ34
16 22 9Ø   SET    R6, $9Ø22$
45         LD     @R5
56         ST     @R6
```

設定R5和R6的指標爲$AØ 34和$9022把$AØ 34的資料搬到$9Ø22 R5, R6各加1。

11.4.6 間接雙位元組載入

```
LDD @Rn   6n
```

Rn指標所指的記憶體位址載入ACC的低位元組，記憶體位址加1載入ACC的高位元組，Rn自動加2，分歧條件依ACC資料設定，溢位旗標清爲零。

例子：

```
15 34 AØ SET R5, $AØ34        $AØ34填入ACC低位元組
65 LDD @R6                    $AØ35填入ACC高位元組
                              R5增爲$AØ36。
```

11.4.7 間接雙位元組儲存

```
STD @Rn   7n
```

ACC低位元組資料塡入Rn所指的記憶體，ACC 高位元組塡入Rn所指記憶體加1。Rn自動加2。分歧條件依ACC設定，溢位旗標清爲零。

例子：

```
15 34 AØ   SET R5, $AØ34 R5, R6放進$AØ34,
16 22 9Ø   SET R6, $9Ø22 $AØ35的資料放進
65         LDD @R5       $9Ø22把$AØ34,
76         STD @R        $9Ø22, $9Ø33, R5,
                         R6都增加2。
```

11.4.8 間接取出

POP @Rn 8n

Rn 指標先減 1，所指的記憶體內容載入 ACC 的低位元組，ACC 的高位元組清為零。分歧條件依ACC 最後的雙位元組設定，所以一定是正數且不為負 1 。溢位旗標清為零。因為Rn 是先減後再做間接取出，所以單位元的堆疊可用 ST @Rn及 POP @Rn 來完成（Rn 為堆疊指標）。

例子：

```
15 34 AØ    SET R5, $AØ34    設定堆疊指標的初值
1Ø Ø4 ØØ    SET RØ, 4        ACC放4
55          ST  @R5          把4放上堆疊
1Ø Ø5 ØØ    SET RØ, 5        ACC放5
55          ST  @R5          把5放上堆疊
1Ø Ø6 ØØ    SET RØ, 6        ACC放6
55          ST  @R5          把6放上堆疊
85          POP @R5          把6從堆疊間接載入ACC
85          POP @R5          把5載入ACC
85          POP @R5          把4載入ACC
```

11.4.9 間接儲存

STP @Rn 9n

Rn 指標先減 1，ACC的低位元組資料存到Rn 所指的位址。分歧條件依ACC的雙位元組設定。STP @Rn 和 POP @Rn 兩個指令都是把資料從高位址到低位址開始搬移，再者STP @Rn 指令可用來完成單位元組堆疊。

例子：

```
14 34 A0   SET R4, $A034     設定指標初值
15 22 90   SET R5, $9022
84         POP @R4           把$A033的資料搬到$9021
95         STP @R5
84         POP @R4           把$A032的資料搬到$9020
95         STP @R5
```

11.4.10 加

ADD Rn An

把Rn的內容加上ACC的內容，低的16個位元放在ACC裏，第17位元放在溢位旗標裏。其餘的分歧條件依照ACC最後值設定。

例子：

```
10 34 76   SET R0, $7634     設定R0, R1初值
11 27 42   SET R1, $4227
A1         ADD R1
A0         ADD R0            加上R1（和爲$B85B，溢位爲零）
                             ACC乘2，爲$70B6，溢位爲1。
```

11.4.11 減

SUB Rn Bn

ACC內容減掉Rn的內容，低的16位元放回ACC裏，第17位元放在溢位旗標，其餘的分歧條件依照ACC最後的值而設定。如果ACC的內容大於等於Rn的內容，則

溢位旗標爲 1 ，反之則爲 0 。 Rn 值不變。

例子：

```
10 34 76    SET R0, $7634    設定ACC(R0及R1的初值。
11 27 42    SET R1, $4227
B1          SUB R1           ACC減R1（差值=$340D，
                             溢位爲1）
B0          SUB R0           ACC清爲零。
```

11.4.12 間接雙位元組堆入

```
POP @Rn   Cn
```

Rn 先減 1 ，Rn 指標所指的記憶體內容放進ACC 的高位元組，Rn 再減 1 。然後所指的記憶體內容放進ACC 的低位元組。分歧條件依照ACC 最後的內容設定。溢位旗標清爲零。因爲Rn 先減再載入 ACC，所以雙位元組的堆疊可用STD @Rn及 POPD @Rn 指令完成（ Rn 爲堆疊指標）。

例子：

```
15 34 A0    SET R5, $A034    設定堆疊指標初值
10 12 AA    SET R0, $AA12    ACC放$AA12
75          STD @R5          把$AA12 放上堆疊
10 34 BB    SET R0, $BB34    ACC放$BB34
75          STD @R5          把$BB34 放上堆疊
C5          POPD @R5         把$BB34 從堆疊上拿下
C5          POPD @R5         把$AA12 從堆疊上拿下
```

11.4.13 比較

CPR Rn Dn

ACC的值與Rn比較，用16位元的減法ACC-Rn，並把低16位元放在R13供以後分歧時測試。如果ACC 大於等於Rn則溢位旗標爲1，否則爲零。除了R13，其餘的暫存器都不改變。

例子：

```
15 34 AØ    SET R5, $AØ34     設定指標
16 BF AØ    SET R6, $AØBF     設定限制指標
BØ          LOOP1 SUB RØ      把ACC清爲零
75          STD @R5           清除二個記憶體

25          LD R5             把R5和R6做比較
D6          CPR R6
Ø2 FA       BNC LOOP1         一直做到溢位爲零。
```

11.4.14 加1

INR Rn En

Rn的內容加1，溢位旗標清爲零，其餘的分歧條件依增加後的值設定。

例子：

```
15 34 AØ    SET R5, $AØ34
BØ          SUB RØ            ACC清爲Ø。
55          ST @R5            清$AØ34爲零。
E5          INR R5            R5增加爲$AØ36。
55          ST @R5            清除$AØ36
                              (不是$AØ35)
```

11.4.15 減1

DCR Rn Fn

Rn的內容減1，溢位旗標清爲零，其餘的分歧條件依照減1後的值設定。

例子：(清除$AØ34起9個位元組)

```
15 34 AØ   SET R5, $AØ34    設定指標初值。
14 Ø9 ØØ   SET R4, 9        設定計數器初值。
BØ         SUB RØ           ACC清爲Ø。
55         LOOP2 ST @R5     清除一個記憶體。
F4         DCR R4           計數器減1。
Ø7 FC      BNZ LOOP2        一直做到計數器爲零。
```

11.5 非暫存器指令

11.5.1 回到6502工作模式

RTN ØØ

控制權交回6502，且從RTN後開始執行起。6502暫存器重新回覆到進入SWEET 16之前，也就是沒有改變。

11.5.2 無條件分歧

BR ea Ø1 d

ea:有效位址，是把下個指令的PC加上d當做新的PC，d是正(bit7=0)或負(bit 7=1)，d是2的補數，從+127到-128，分歧指令對旗標無影響。

注意：分歧的有效條件計算方法與6502的相同。可用APPLE Ⅱ監督程式裏的單位元加、減副程式計算d。

```
d = $8Ø   ea = PC + 2 - 128
d = $81   ea = PC + 2 - 127

d = $FF   ea = PC + 2 - 1
d = $ØØ   ea = PC + 2 + Ø
d = $Ø1   ea = PC + 2 + 1

d = $7E   ea = PC + 2 + 126
d = $7F   ea = PC + 2 + 127
```

例子：

```
$3ØØ:  Ø1 5Ø   BR $352
```

11.5.3 無溢位(爲零)則分歧

```
BNC ea Ø2  d
```

只有在溢位爲零時才執行分歧，否則繼續往下執行。分歧指令對旗標無影響。

11.5.4 溢位則分歧

```
BC ea Ø3  d
```

只有在溢位爲1時才執行。

11.5.5 正數則分歧

```
BP ea Ø4  d
```

上個指令的結果爲正數或上次轉移的資料爲正數，則執行分歧，若爲負數則繼續往下執行。

例子：（$AØ34到$AØ3F清爲零）

```
15 34 AØ   SET R5, $AØ34   設定指標初值。
14 3F AØ   SET R4, $AØ3F   設定上限。
BØ         LOOP 3  SUB RØ  ACC清爲零。
55                 ST @R5  清除記憶體，並把R5加1。

24                 LD R4   把指標和上限比較。
D5                 CPR R5
Ø4 FA          BP  LOOP3   一直做到完成。
```

11.5.6 負數則分歧

```
BM ea Ø5  d
```

上個指令的結果或者上次轉移的資料爲負，則執行分歧。

11.5.7 爲零則分歧

```
BZ ea Ø6  d
```

上個指令的結果爲零則分歧。

11.5.8 不爲零則分歧

```
BNZ ea Ø7 d
```

上個指令的結果不爲零則分歧。

11.5.9 負 1 則分歧

```
BM1 ea Ø8 d
```

上個指令的結果爲負 1 （ $FFFF ）則分歧。

11.5.10 非負 1則分歧

```
BNM 1 ea Ø9 d
```

上個指令的結果不爲負 1 則分歧。

11.5.11 中斷

```
BK ØA
```

執行 6502 BRK 指令。若想在 BRK 後繼續執行 SWEET 16 的指令，則須在重新進入之前把堆疊指標修正爲執行 BRK 之前的指標。

11.5.12 從 SWEET 16的副程式返回

```
RS ØB
```

RS 結束 SWEET 16 副程式，並回到 SWEET 16 呼叫的程式，往下繼續執行（還在 SWEET 16 工作模式）。R12 是 SWEET 16 副程式返回堆疊的指標，將被減 2 。分歧條件不改變。

11.5.13 呼叫 SWEET 16的副程式

```
BS ea ØC  d
```

分歧到 PC+2+d 的位址繼續 SWEET 16 的工作模式，現在 PC 被放在“SWEET 16 副程式返回位址”，堆疊，其指標爲 R12 ，且 R12 自動加 2 。溢位旗標清爲零，且分歧條件被設定爲 ACC 內容的情形。

例子：（呼叫一個“搬記憶體”的副程式把 \$AØ34 ～ \$AØ3B 搬到 \$3ØØØ ～ \$3ØØ7 ）

```
15 34 AØ   SET R5, $AØ34    設定指標1
14 3B AØ   SET R4, $AØ3B    設定上限
16 ØØ 3Ø   SET R6, $3ØØØ    設定指標2
ØC 15      BS  MOVE         呼叫副程式
45 MOVE    LD  @R5          搬一個位元組
56         ST  @R6
24         LD  R4
D5         CPR R5           測試做完沒有
Ø4 FA      BP  MOVE
ØB         RS               返回呼叫程式
```

11.6 工作原理

呼叫 SW 16 即開始執行 SWEET 16 的工作模式，所有 6502 的暫存器同時也都被保存起來，而且在執行 SWE-

ET 16的RTN指令後，返回6502工作模式時再度取回所有暫存器。如果使用者不在乎暫存器內容改變，可以呼叫SW16＋3以進入SWEET 16工作模式，可以節省大約30 μ秒。因爲這樣可能由16進位變爲10進位的工作方式。最好還是呼叫SW16以進入SWEET 16。

把6502暫存器存起來之後，SWEET 16將它的PC設定爲呼叫指令的下一個碼，（把6502的堆疊上拿下二個位元組，此二位元組爲呼叫指令的返回位址。）所以緊跟著呼叫指令的就是SWEET 16的單、雙、三位元組的指令，如同6502一樣按順序排下去。SW16B裏的主程式不斷地呼叫"執行指令"程式以執行SW16的程式。

SW16C副程式則把PC(R15)不斷地增加，並且提取下個運算碼，若不是爲暫存器指令就是非暫存器指令。假設爲暫存器指令，就把低4位元乘2之後放在X暫存器裏，當作索引(INDEX)。然後再決定指令型式。暫存器指令把所指定的暫存器位址存在R14的高位元組，當作"上一次指令所用的暫存器"，讓以後的分歧指令用做測試條件。非暫存器指令則把右邊的半位元組當作它們的運算碼，把PC加1指到存放位移的位址，"上次指令所用暫存器"的索引放到6502的A—暫存器，來測試分歧條件，再把Y—暫存器清爲零。

11.7 RTS如何成爲JSR？

每一種指令型式都有一個相對的副程式，這些副程式的進入位址都建立成爲一個表，直接用運算碼做爲索引，並把

所有進入點的高位元組都設定在同一頁之中，所以每個進入點在表中只佔一個位元組。6502 間接跳躍的指令可把控制權交到正確的副程式中：

```
LDA    #ADRH      高位元組
STA    IND+1
LDA    OPTBL,X    低位元組
STA    IND
JMP    (IND)
```

為了節省程式空間，先把進入位址（減 1 ）放上堆疊，然後執行一個 6502 的RTS（副程式返回），把進入點放進 6502 的 PC 裏(先加 1 ），其結果就是跳到副程式去執行。

11.8 運算碼的副程式

暫存器指令的副程式用到 6502 的兩種定址方式：用X做索引，用X做索引後間接定址，以拿到指定的暫存器或做為間接資料。大部份暫存器指令的結果就留在暫存器裏，且可做為下個分歧指令的條件，因為所用到的暫存器號碼已留在R14 的高位元組裏。ADD和SUB 指令暫存器號碼為RØ，而CPR指令則為R13 。

通常R13的高位元組存的是上個指令暫存器號碼，而低位元組則為零。如果ADD, SUB, CPR指令產生溢位則設定低位元組。

SET指令把PC加 2，並取得二個位元組的資料放進暫存器裏。依照6502 的方式，先是低位元組，然後跟著高位元組。

大部份非暫存器指令都是分歧指令，對應的副程式決定“上個指令的結果”是否符合分歧的條件，如果符合，則把SWEET 16的PC加上下一個位元組的位移值（+127或−128）。

RTN指令先把65Ø2暫存器原來的值填回暫存器，把副程式返回位址從堆疊上取下，並從SWEET 16的PC執行間接跳躍，如此則控制權又回到65Ø2，從RTN下個指令執行起。

BK指令則眞正的執行65Ø2的BRK 指令，把控制權交到中斷處理程式。

SWEET 16的呼叫程式指令BS及副程式返回指令（RS）可以有任意的層次，使用者首先須設定R12的初值做爲堆疊指標，並且不得破壞R12。

11.9 記憶體配置

SWEET 16所須的記憶體爲零頁32個連續的位元組，做爲SWEET 16的暫存器，4個位元組以儲存65Ø2的暫存器，及一些65Ø2副程式呼叫所須的堆疊。如果你不須保留65Ø2 的暫存器，則可去掉SAVE及RESTORE兩個副程式及呼叫此兩副程式的指令。如此可省下四個零頁位址：ASAV，XSAV，YSAV，PSAV。

11.10 使用者自行修改

你或許想增加你自己的指令到SWEET 16現有的指令

群，如果你使用未定義的運算碼 $ØE與 $ØF ，要記得SWEET 16 把此兩個設定為雙位元組指令。你或許想用 65Ø2 BRK 指令做為進入SWEET 16 工作模式的指令，如此則每次進入 SWEET 16 都可節省 2 個位元組。或者你想用 SWEET 16 BK指令當作呼用“ CHAROUT”，而在中斷處理程式裏執行“ 輸出字母 ”。在SWEET 16 裏可做絕對跳躍，只要把你所想跳到的位址減 1 放在RØ ，然後執行 ST R15 指令即可。

果粉藏書

12. APPLESOFT 的原始程式

12.1 SOURCEROR.FP

MERLIN 系統磁碟片的反面有一個程式 SOURCEROR.FP可產生APPLESOFT 的原始程式，並且有完整的標題及註解。

此程式首先把你的APPLE Ⅱ上的APPLESOFT 掃瞄一次，把標題及註解加入後，產生四個文字檔：T.APSOFT I， T.APSOFT II, T.APSOFT III, T.APSOFT Ⅳ。

爲了節省空間，這四個文字檔使用大量的集體指令，這些集體指令都在本磁片另外一個檔：APPLESOFT.S.裏定義。使用 PRTR 指令，開始組合這些檔就可以印出非常整齊的APPLESOFT 原始程式。詳細的操作方法將在後面說明。

請特別注意：這份原始程式並不是“正式的”原始程式，不是APPLE 公司所出版的，而是作者自己研究、解釋APPLESOFT的僅讀記憶體(ROM)所做的一份說明。APPLE公司絲毫未參予此份資料的出版，出版前也未經過APPLE公司校正過。使用時不可將它視爲APPLE 公司所贊助的。

再者：本公司(SOUTHWESTERN DATA SYSTEM)不對這份資料做任何正確性及可用性的保證，提供它只是做爲MERLIN使用者一個額外消遣罷了。

警告：SOURCEROR.FP及其一些暫時的工作檔案無法用CATALOG指令看到，且BRUN SOURCEROR.FP之後就被刪除掉了。爲此，你必須先用COPYA把MERLIN反面磁片先做備份，再用新COPY的磁片依照下面步驟做。

12.1.1 產生Apple Soft的組合原始程式的步驟 1

1. BRUN SOURCEROR.FP ，用你的備份磁片(看上面的警告)。
2. 重新開機載入MERLIN, L)oad (載入) APPLESOFT。
3. RAM卡版：在編輯模式下，設定 SYM爲$8ØØØ ，打入PRTR指令，並開始組合(ASM)此檔。螢幕將出現：

```
:SYM $8ØØØ
:PRTR 1 "I8ØN"APPLESOFT LISTING"
:ASM
```

4. 48K版：在編輯模式下，設定HIMEM與SYM 爲$4EØØ 輸入PRTR命令，然後組合(ASM)該檔案。螢幕將會出現：

```
:HIMEM:$4EØØ <—注意：分號
:SYM $4EØØ
:PRTR 1 "CTRL-I8ØN"APPLESOFT LISTING"
:ASM
```

照上面的例子，PRTR指令會送“<CTRL-I>80N”到印字機界面卡，並會在報表紙最上一行印出“APPLE-SOFT LISTING”。

MERLIN會發出以下的問題：“GIVE VALUE FOR SAVEOBJ:”這是問你要不要產生目的碼並存回磁碟，建議你最好回答“ Ø ”，如此做之後就會開始列印。如果你回答“1 ”，則爲了把目的碼存回磁碟，會使速度降低。產生目的碼的用處是好讓你做比較。

MERLIN現在會做一些預防檢查，確定一切都OK之後才會開始列印。磁碟機將會轉動數次，之間會隔相當的時間，這是正常的，整個檢查程序大概須3.5分鐘。

然後MERLIN開始列印一份非常詳細的APPLESOFT原始程式，包括標題及註解，總共須紙105頁（包括符號表），幾乎要一個半小時才完成列印。（每秒80字的印字機）。

13. 語彙解釋

ABORT	一終結、中斷，提早結束操作。
ACCESS	一取、拿，尋找或讀取資料。
ADDRESS	一位址。
ALGORITHM	一演譯法，一種解決問題的方法。
ALLOCATE	一保留空間。
ASCII	一美國國家標準交換碼。
BASE	一基底。
BINARY	一二進位。
BIT	一位元。
BRANCH	一分歧，從一個新的位址繼續執行。
BUFFER	一緩衝器，資料暫時存放的位址。
BYTE	一位元組， 8 個位元。
CARRY	一溢位、65Ø2 微處理機的一個旗標。
CHIP	一晶片、積體電路裏的矽晶片。
CODE	一碼。
CTRL	一控制， CONTROL 的簡寫。

CURSOR	—游標，一個閃爍的空白字元，指示下一個等待輸入的位址。
DATA	—資料。
DECREMENT	—遞減。
DEFAULT	—假設值。
DELIMIT	—分隔，例如 BASIC 裏的“ : ”。
EQUATE	—等於，設定變數值。
EXPRESSION	—算式，表示式。
FETCH	—取、拿。
FIELD	—欄，資料特定的部份。
FLAG	—旗標，暫存器或記憶體用以表示某個條件的眞僞。
HEX	—16 進位，由 Ø～9，A～F的字母所組成的 16 進位數系。
HIGH-ORDER BYTE	—高位元組，雙位元組中較高效的位元組。
HOOK	—開關。指向 I/O 位址的指標。
INCREMENT	—遞增。
INITIALIZE	—起始設定。
I/O	—輸入／輸出。
INTERFACE	—界面。
INVERT	—反向。
LABEL	—標題，一個位址或變數的名稱，通常依照其意義而命名。
LOOKUP	—查表。
LOW-ORDER BYTE	—低位元組，雙位元組中較低效

	的位元組。
LSB	—最低效位元，或位元組。
MACRO	—集體指令，在組合器裏，用一個名稱以呼叫一段碼插入目的碼之中。
MICROPROCESSOR	—微處理機，微電腦的心臟。（在 APPLE 裏是 65∅2 ）。
MOD	—除～的餘數。
MODE	—模式，某種特定的操作方式。
MODULE	—模組，程式中特定功能的一段程式。
MNEMONIC	—助憶符號，某個字的縮寫以幫助記憶。
MSB	—最高效位元。
NULL	—無效的。
OBJECT CODE	—目的碼，組合器所產生的碼。
OFFSET	—位移值。
OPCODE	—運算碼，6502 執行的指令。
OPERAND	—運算元。6502 執行的資料。
PAGE	— 256 位元組的記憶體區域。
PARAMETER	—參數，程式中用到的常數或變數。
PERIPHERAL	—週邊裝置。
POINTER	—指標。某個記憶體位址，包含的內容指到記憶體其他位址。
PORT	—埠，週邊裝置實際連接的位址。

PROMPT	一等待輸入符號。
PSEUDO	一虛擬的。
RAM	一隨意存取記憶體。
REGISTER	一暫存器。
RELATIVE	一相對的。
ROM	一僅讀記憶體。
SIGN BIT	一正負位元，位元組的第七位元，如位元組大於 $80 則爲負。
SOURCE CODE	一組合器所須的輸入檔，組合後可產生目的碼。
STACK	一堆疊，一種先進後出的貯存位址。
STRING	一字串。
SWEET 16	一一個模擬 16 位元微處理機的程式。
SYMBOL	一符號。
SYNTAX	一語法。
TABLE	一表。一串的位元組，雙位元組，被程式所用。
TOGGLE	一反覆，由某種狀態轉移到另一個狀態。
VARIABLE	一變數。
VECTOR	一向量。所須的位址或分歧的位址。

14. 程式範例

第一組三個程式範例是由 Steve Wozniak 和 Allen Baum 所寫的，非常精彩的組合語言程式。還可以在整數培基 F4 僅讀記憶體上找到。在系統磁片上可以看到此三個程式，都是原始程式的格式，以方便 APPLE Ⅱ + 的使用者。(APPLE Ⅱ + 上沒有整數培基)。這三個程式可以放在記憶體中任何位置。

14.1 浮點運算

此浮點運算爲單精密度的程式，可以在 BASIC 或組合語言裡呼叫。原始程式中就有詳細說明其用法。

14.2 乘／除運算

此程式是給組合語言程式做爲副程式，可做 4 位元組的乘除。其用法請看原始程式。磁片上還有另一程式範例說明除法的運用，是由 Dave Garson 所寫。

14.3 PRDEC (顯示10進位數)

這是最常被用到的程式。可以把2進位數(Ø～65535)換成十進位並且輸出到顯示器上。能夠很容易的在組合程式中運用。先把高位元組放在A暫存器。低位元組放在X暫存器，然後呼叫PRDEC。或者把高位元組放在$F3，低位元組放在$F2，然後呼叫PRDEC＋4。

14.4 MSGOUT (顯示資料)

此程式是由Andy Hertzfeld所寫，把ASCII的字串用組合語言送出去顯示。如果使用MERLIN的INV或FLS虛指令和MSGOUT混合使用則須把ORA #$80 這一列刪掉，而正常的ASCII碼須設定bit 7為1。在同一個原始程式中有二個副程式，接收使用者的輸入ASCII碼及16進位碼。

14.5 UPCON (轉為大寫)

由Glen Bredon所寫，把原始程式中註解的小寫字母轉成大寫字母。讓沒有顯示小寫的APPLE Ⅱ也能看到註解。先載入原始程式，然後在執行模式下BRUN UPCON。

14.6 GAME PADDLE Printer Driver

早期當APPLE Ⅱ剛發展出來時並沒有印字機界面，甚

至從未考慮到要有印字機。但APPLE 公司發展程式時必須把程式HARD COPY出來，所以就用GAME PADDLE 接串聯的印字機。由Randy Wiggington和Steve Wozniak 所寫。發表在“紅皮書”使用手冊中。之後Disk Ⅱ發展出來時並不能工作，因爲此程式不管DOS的存在，設定自己的I/O開關。然後Aldrich兄弟解決了問題。在目前當然沒有必要用到遊戲控制桿界面，印字機界面卡已發展出來，且工作的很好。

系統磁片上此原始程式有完整的註解說明。可做爲示範，讓寫印字機界面的程式師參考。

15. 公用程式

15.1 FORMATTER 定格式

此程式加強 MERLIN 的功能，讓它能做爲一般文字編輯系統。它自動把檔案分段爲每列若干字。在原來的檔中每段以一空列分開。

要使用 FORMATTER，首先在 EXEC 模式下 BRUN FORMATTER，把自己載入到 $9Ø64，然後再搬到 $94AØ，如此可以方便使用者建立 USER 指令向量，只要利用 USER 命令，即可把記憶體內的檔案定格式。

FORMAT 先詢問要分段的位址區域，如果只回答一個位址則從此位址到檔結束都會被分段。再詢問每列的長度，必須小於 25Ø。最後詢問此檔是否要左右都對齊（不然只有左邊對齊）。

此程式一開始先檢查每一列是否以空白開頭，如果不是則在每一列首插入空白，這是讓 TAB 指令有個左限，然後再用 PRINT 印出。

爲了要左右兩邊都對齊，FORMATTER 在需要插入空白的地方插入反向空白，爲的是能夠很容易的定位與移走，以

方便下一次 FORMATTER。注意：FORMAT 之後的檔不可用 TEXT和FIX指令（除非此檔另有備份）。對於外來的原始程式，最好先用FIX指令以確定此檔的格式符合FORMATTER 的要求。爲了同樣的原因，在編輯之前最好只有左邊對齊。

別忘了在印出已定格式過的檔案之前用TABS命令。

15.2　CHRGEN 70

CHRGEN7Ø 是讓使用者能在高解像度圖區顯示 70 個字 24 行的顯示字幕。因爲解像度提高，而普通電視的解像度不夠，所以需要用CRT顯示器才能得到滿意的效果。

要使用CHRGEN 7Ø，首先在 MERLIN 執行模式下 BRUN CHRGEN 7Ø，之後會自動把程式起點設定爲 $4ØØ1，並且清除記憶體裏的所有檔案。只要用 USER 命令就可以進入高解像度圖區顯示。

要跳出CHRGEN 7Ø只要在編輯模式下打入 VID Ø，VID 16 或 PR#Ø。由編輯模式進入執行模式會自動離開CHRGEN 7Ø，要重新進入只要在編輯模式下用USER命令即可。如爲了容納較長的原始程式，需要永遠的跳出CHRGEN 7Ø，這時你必須重新BRUN MERLIN。

在 CHRGEN 7Ø裏使用PRTR命令，只要利用 PRTR8“檔名”即可。如果別的USER命令重新蓋掉USER的向量，可用重新 BRUN CHRGEN 7Ø，或進入監督程式執行 9ØØG 以重新設定 CHRGEN 7Ø 的向量。後面的方法當然是假定CHRGEN 7Ø 仍然還在記憶體裏。

CHRGEN 7Ø裏包括FORMATTER程式，要在CHRGEN 7Ø裏執行FORMATTER只要在編輯模式下按CTRL-T。注意：除非已列印過否則此命令將沒有作用。

CHRGEN 7Ø同時包括一些鍵盤集體指令，ESCAPE 之後跟著一個按鍵就會產生鍵盤集體指令。已定義完成的集體指令爲：

```
* > " ' # +
: . 2 7 3 ; X Y D H P @ L S - O C A E
```

此集體指令表位於CHRGEN 7Ø程式最後面，$15ØØ，可以修改。但必須以$FF做爲結束。

注意：在CHRGEN 7Ø使用中時不可以載入一個太大的xxx.s原始檔（不得大於88磁區SECTOR），否則將會蓋到DOS的緩衝區，使得系統當掉。文字檔則不受此限。

15.3 XREF, XREF.XL,和STRIP

XREF，XREF.XL和STRIP公用程式提供一個便捷的方法由原始檔案中產生“相互對照表”（Cross-Reference Listing），這種對照表可以幫助你很快的在程式中找到使用某標題的指令位址。當爲大型程式除錯（Debug）時相互對照表更顯出其重要性。

MERLIN組合器能在組合結束時印出符號表，雖然此表告訴你某個標題的值，但卻不知程式中那裏使用此標題。XREF，XREF.XL和STRIP就是補其不足。

XREF可以印出按字母或數字排列的相互對照表，大約可處理1ØØØ到2ØØØ列之間原始程式中的所有符號的使用

情形。

XREF.XL與XREF 功能差不多，但能處理3～4倍的原始程式，它把對照表先寫入磁碟，而後再從磁碟叫出送到印字機。

STRIP把原始程式中所有的註解去掉，以縮減原始程式的大小。

15.3.1 MERLIN 符號表列印範例

符號表——按字母順序排列

ADD	=$F786	BC	=$F7BØ	BK	=$F7Ø6

符號表——按數字順序排列

BK	=$F7Ø6	ADD	=$F786	BC	=$F7BØ

15.3.2 MERLIN XREF 列印範例

相互對照表——按字母順序排列

ADD	=$F786	1Ø1	185*
BC	=$F7BØ	9Ø	2Ø7*
BK	=$F7Ø6	1Ø4	121*

相互對照表——按數字順序排列

BK	=$F7Ø6	1Ø4	121*
ADD	=$F786	1Ø1	185*
BC	=$F7BØ	9Ø	2Ø7*

由上面的例子（SWEET 16的原始程式中的標題）可看出標題的值是用“＝”表示，原始程式中出現標題的列號

列在後面，打 * 號的那個列號是此標題定義的地方。

此外對於用 PUT 指令叫進來的檔案其列號則是用“134.82 ”表示，其意義爲主程式第 134 列中的 PUT 指令所叫進來檔的第 82 列。

15.3.3 XREF 使用說明

1. 進入 MERLIN 執行模式，確定你正在使用的檔已存好。選擇有 MERLIN 的那一個磁碟機號碼。
2. Catalog 後，當 MERLIN 要求輸入 DOS 命令時，輸入：BRUN XREF（記憶體中的檔會被清除掉)。
3. 按 <CTRL-C><RETURN>，然後載入你的檔案，起動你的印字機（用 PR# 或 PRTR 命令），XREF 是專爲印字機設計的。
4. 輸入正確的 USER 命令：

USER Ø－列印原始程式及按字母排列的相互對照表，（USER 等於 USERØ ）。

USER 1－列印原始程式及按字母和數字排列的兩個相互對照表。

USER 2－只列印按字母排列的相互對照表。

USER 3－只列印兩種相互對照表。

USERØ ～ 3 的命令不會將條件組合爲 OFF 裏的標題列印在相互對照表裏。

和 USERØ ～ 3 一樣，USER4 ～ 7 命令則列印所有的標題到相互對照表裏。

注意：你可任意的改變 USER 命令，直到打入 ASM 命令。

5. 輸入ASM命令開始組合，並列印。

15.3.4 使用 XREF 時注意事項

XREF在組合第二階段輸出的列表中找出標題，建立一個相互對照表，位址由HIMEM往上（不是目的碼而是對照表），如果目的碼是DSK命令產生的，則還可在磁碟中產生目的碼。此對照表中每一個標題須用掉6個位元組，因此很快就會把記憶體用完，所以對大檔案必須把HIMEM儘可能的降低。（W∅ 命令可以告訴你程式結束位址，這位址就是HIMEM可設定的最低位址）。

因為XREF須要組合器輸出；LST OFF 後的標題將不會出現，同時也不會出現在相互對照表裏。尤其重要的是：組合結束時LST ON 是必須的，XREF才能正常操作。同樣的原因，也不可以按CTRL－D。

集體指令須特別考慮，因為其語法變得相當複雜，XREF可能會搞糊塗，而停止組合。這種情形通常發生在＞＞＞（PMC）虛運算碼之後跟著文字串或括號；而且又利用EXP OFF停止集體指令的擴充列表。你可以首先利用一般的方法（以LST OFF停止產生符號）印出組合列表，然後利用USER2 或 3以EXP ON 印出相互參考表，以解決以上的問題。

使用集體指令另外要小心的還有：集體指令中定義的標題其不具有完整意義的不會被放到相互對照表裏：

```
DEF       MAC                      <---定義集體指令
          CMP   #]1
          BNE   DONE
          ASL
DONE      <<<
------------------------           <---程式起始
          >>>   DEF.GLOBAL         <---呼叫集體指令
```

上面的例子GLOBAL會在相互對照表裏，但DONE則否。

15.3.5 XREF.XL 使用說明

XREF.XL 是用來處理3到4倍大的檔案（跟XREF比較），原先是設計來產生APPLESOFT培基的相互對照表，它幾乎是XREF.XL所能處理最大的檔案了。

依照XREF的5個步驟使用XREF.XL，只要在第2步把XREF改為XREF.XL即可。XREF.XL工作方式和XREF相類似，只是把相互對照表寫到磁碟去成為X.R.FILE（執行完後可以把X.R.FILE 刪掉）。組合結束後再把X.R.FILE 載入記憶體，蓋掉原始程式（$9Ø1 起），如步驟1所說，先把你的原始程式存到磁碟。

15.3.6 使用 XREF.XL 注意事項

——原始程式在結束之後將會從記憶體內消失，如上所述，首先確定已存好。

——使用XREF.XL時考慮用空白磁片，因為X.R.FILE可能會很大。

——相互對照表檔X.R.FILE 是寫在最後用到的那個

磁碟機上，如果你的原始程式有PUT命令，須確定XREF.XL可以找到其餘的原始程式檔。可以把這些檔搬到空白磁片上或者在PUT命令中設定磁碟機SLOT及DRIVE。

——不像XREF，XREF.XL不受HIMEM設定的影響，然而在建立相互對照表時，會檢查是否能放到記憶體裏(由$901到SYM或者到$9853)，如太大則會有OUT OF MEMORY的錯誤信息。

——XREF.XL若在原始程式中找到DSK命令，則會停止執行並發出ILLEGAL DSK ATTEMPTED的錯誤信息。可以把DSK命令改為註解以保持列數相同。

產生APPLESOFT相互對照表步驟說明

1. 先用STRIP把所有APPLESOFT的檔去掉註解後，存到空白磁片上。
2. 在主檔APPLESOFT.S(STRIP後)把EXP OFF改為EXP ON(或去掉)。
3. 把SAVOBJ KBD這行及上下的DO-FIN迴圈去掉。
4. 確定最後使用的磁碟機已裝了STRIP後的檔。
5. 輸入PRTR命令，然後用USER2或USER 3命令。
6. 打入ASM，並開始列印一個大約25頁的相互對照表。

15.4 STRIP 去掉註解

非常大的原始程式中，或者註解非常多的程式中，很容易在產生相互對照表時發生記憶體不夠的錯誤。

公用程式 STRIP 可以讓 XREF 程式處理兩倍以上的原始程式。

依照下面的步驟使用 STRIP：

1. 確定先把有註解的檔做個備份，然後載入記憶體中。
2. 進入編輯模式，在程式最後加個 LST OFF，然後 ASM。
3. 把剛剛加的 LST OFF 去掉（重要！）。
4. 離開編輯模式，選擇磁碟機，然後 CATALOG，輸入 BRUN STRIP，就會把記憶體裏原始程式的註解都去掉。
5. 按 <CTRL-C><RETURN>進入編輯模式，再用 XREF 或 XREF.XL 以產生相互對照表。

15.5 PRINTFILER

PRINTFILER 是 MERLIN 系統磁片上的公用程式，把組合後的列表輸出存到一個循序文字檔（Sequential Disk File），同時也能讓你選擇做“檔案壓縮”以節省空間，並選擇把螢幕輸出關掉，以加速操作。

PRINTFILER 產生的文字檔包括目的碼（這是一般原始程式所沒有的）；可以完全的顯示組合語言程式與目的碼，而不必為了看目的碼每次都要組合一次。

15.5.1 應用

應用包括：

——利用文件處理器（ Word processor ）所整理出來以產生完整的說明。

——把此文字檔利用調變器（ modem）經由電話線傳送。

——把此檔寄給某人，他需要完整的檔案而不用組合程式。（例如雜誌的編輯等等）。

15.5.2 如何使用 PRINTFILER

1. 從執行模式中，先把你的原始程式存回磁碟中，再 BRUN PRINTFILER 。
2. 按（ RETURN），把你的原始程式由磁碟中載入。
3. 跳出編輯模式，選擇你要存文字檔的磁碟（用Drive ），進入編輯模式，打入USER“檔名”，（括號也要打），須注意若以後要用Read 把此文字檔讀入，則須在“檔名”之前加上“ T. ”，例如“ T.DUMMY”，而不是“ DUMMY”。
4. 打入ASM，回答“ UPDATE SOURCE：”後PRINTFILER會在磁碟中產生一個文字檔。在螢幕上什麼都看不到，因爲PRINTFILER 已設定不輸出到螢幕。

15.5.3 修改 PRINTFILER

PRINTFILER 有兩個地方可以修改：檔案壓縮及螢幕輸出，再者改變可以是暫時性的或永久性的。

檔案壓縮是把存到磁碟的文字檔裏大量的空白轉換成一個字元，字元的最高位元爲 0 。在列印壓縮過的檔案時，壓縮空白以反向字元表示（反向的A＝1 個空白，反向的B＝

2 個空白，反向的 C = 3 個空白），依此類推。

不壓縮則照多少空白就存多少個位元組。雖然這個檔無法 ASM，但可讀性甚高，壓縮的檔可用 Southwestern Data System 另一產品" DOUBLETIME PRINTER"可把它印出來。

螢幕輸出就是把送到磁碟的文字檔也顯示在螢幕上，但如此會使得速度變慢。

15.5.4 PRINTFILER 的比較

原始檔：SWEET 16.S（21 磁區）

檔案型態	螢幕	磁區	時間
未壓縮	輸出	61	59秒
未壓縮	不輸出	61	44秒
壓縮	輸出	38	42秒
壓縮	不輸出	38	30秒

如你所看到，螢幕不輸出使得 PRINTFILER 加快約 25 %，檔案壓縮也會增加速度。

再者未壓縮的文字檔約為壓縮的文字檔的二倍，約為原始程式檔的三倍。

15.5.5 修改 PRINTFILER 的選擇

暫時性的修改：

進入編輯模式，打 MON，然後輸入 $300:

```
  300:00 00  檔案壓縮，螢幕不輸出
或300:00 80  檔案壓縮，螢幕輸出
或300:80 00  檔案不壓縮，螢幕不輸出
或300:80 80  檔案不壓縮，螢幕輸出

        (正常值爲 300:80 00 )
```

按<RETURN><CTRL-Y><RETURN>回到編輯模式，你打入的值一直有效，直到你再次BRUN PRINTFILER。

永久性的修改：

1. 載入PRINTFILER，打入ASM，它會問以下的問題。
2. 回答“UPDATE SOURCE?”後，再問“GIVE VALUE FOR FORMAT:”，如果回答“Ø”，則檔案壓縮，若按“1”則檔案不壓縮。
3. 再問“GIVE VALUE FOR MONITOR:”，如果按“Ø”，則螢幕不輸出，如果回答“1”則螢幕輸出，之後PRINTFILER開始組合爲目的碼。
4. 離開編輯模式，按“O”存目的碼，接著再BRUN這目的碼，它就依照你所修改的情形執行。所以可以有不同版本的PRINTFILER，以代替設定不同值。

Retro Workshop 果粉工作室

HK$ 28.00